VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ OPTIMALIZACE TURNUSŮ JÍZDNÍCH ŘÁDŮ

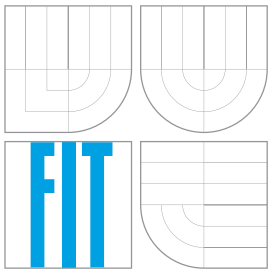
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB FILÁK

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ OPTIMALIZACE TURNUSŮ JÍZDNÍCH ŘÁDŮ

EVOLUTIONARY OPTIMIZATION OF TOUR TIMETABLES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. JAKUB FILÁK

AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ JAROŠ

SUPERVISOR

BRNO 2009

Abstrakt

Tato práce se zabývá problémem optimalizace turnusů jízdních řádů. Obsahuje popis jízdních řádů se zaměřením na popis turnusů a jejich optimální tvorby, jak s pomocí klasických metod, tak s využitím heuristik. Dále jsou popsány evoluční algoritmy, které přímo souvisejí s prací. Důraz je kladen na popis genetických algoritmů a metody zakázaného hledání. Na základě poznatků jsou dále navrženy operátory křížení a mutace a lokální vyhledávací metoda pro memetický algoritmus řešící tvorbu optimálních turnusů jízdních řádů. V souladu s návrhem algoritmů jsou analyzovány požadavky na optimalizační systém. Práce obsahuje popis implementace analyzovaného systému a diskutuje výsledky experimentů se systémem.

Klíčová slova

Optimalizace, Evoluční algoritmy, Jízdní řád, Turnusy, Zakázané hledání, Memetické algoritmy

Abstract

This thesis deals with the problem of vehicle scheduling in public transport. It contains a theoretical introduction to vehicles scheduling and evolutionary algorithms. Vehicle scheduling is analyzed with respect to the bus timetables. Analysis of evolutionary algorithms is done with emphasis on the genetic algorithms and tabu-search method After the theoretical introduction, a memetic algorithm for the given problem is analyzed. Finally, the thesis contains a description of the optimization system implementation and discusses the experiments with the system.

Keywords

Optimalization, Evolutionary algorithms, Time tables, Tabu search, Memetic algorithms

Citace

Jakub Filák: Evoluční optimalizace turnusů jízdních řádů, diplomová práce, Brno, FIT VUT v Brně, 2009

Evoluční optimalizace turnusů jízdních řádů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jiřího Jaroše. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Jakub Filák

25. května 2009

Poděkování

Děkuji vedoucímu Ing. Jarošovi za pomoc při tvorbě této práce. Dále děkuji kolegovi Ing. Jiřímu Doležalovi za poskytnutí informací o jízdních řádech. V neposlední řadě bych rád poděkoval firmě UNIS Computers, s.r.o za čas a zdroje, které mi poskytla na vytvoření této práce.

Obsah

Kapitola 1

Úvod

Již od dávných dob lidé cestují. Dříve cestovali za poznáním a dobrodružstvím, dnes je cestování více méně každodenní nutností. Se zvyšujícím se počtem cestujících bylo také potřeba více dopravních prostředků. S vetším počtem dopravních prostředků bylo nutné vytvořit systém jejich kooperace při obsluze požadavků cestujících. Systémy dopravních prostředků se časem staly velmi složité. Aby cestující věděli jak se dostanou na místo cesty byly vytvořeny jízdní řády, které definovaly čas a místo odjezdu dopravního prostředku. Jízdní řády jsou vytvářeny pro všechny druhy dopravy a pro každý z nich mají speciální formát. Tato práce se bude zabývat jízdními řády pro silniční linkovou dopravu, pro niž se sestavují linkové jízdní řády.

Pro sestavování jízdních řádů jsou dnes vypracovány složité postupy ověřené praxí. Většina jízdních řádu je v dnešní době sestavována manuálně s částečným využitím výpočetní techniky. Nejčastějším způsobem sestavení jízdního řádu je provedení analýzy a poté ruční zapsání všech informací, například do databáze podnikového informačního systému. V tomto postupu nejsou prováděny žádné větší optimalizace. Hlavní příčinou proč se optimalizace neprovádějí je obrovská složitost modelů popisujících tvorbu jízdních řádů. Dále je to způsobeno velkým počtem kritérií, na které se musí při tvorbě brát ohled[2].

Celková optimalizace jízdních řádů je velmi složitá a jde o hudbu budoucnosti. S příchodem výkonnější výpočetní techniky bude jistě možné zadat stroji několik informací o oblasti, pro kterou je třeba vytvořit jízdní řád, a během několika minut bude vytvořen optimální jízdní řád. Dnes se však můžeme zaměřit na optimalizaci jen některých částí jízdního řádu. Jednou z možných částí pro optimalizaci jsou turnusy jízdních řádů. V této části je potenciál pro uspoření milionů korun na provozu dopravní společnosti. Turnusy jízdních řádů jsou v podstatě časové rozvrhy pro dopravní prostředky. Na základě těchto rozvrhů dopravní prostředky obsluhují spoje linek. A právě optimalizací vytváření těchto časových rozvrhů se bude práce zabývat.

Jako nástroj optimalizace budou použity některé evoluční algoritmy. Existují sice klasické analytické postupy, popisující jak sestavovat turnusy jízdních řádů a přitom je vytvářet optimální. Avšak tyto matematické postupy většinou znamenají prozkoumání téměř celého

prostoru možných řešení. Mimo tyto matematické postupy se dodnes praktikuje manuální postup. Každý z přístupů má své výhody i nevýhody. Hlavní výhodou matematických postupů je nalezení optimálního řešení i když za cenu neúnosné časové délky hledání. Evoluční algoritmy naproti tomu nezaručují nalezení optimálního řešení za každé situace. Jejich hlavní výhodou je menší časová náročnost pro velmi složité úlohy, což sestavení optimálních turnusů je, a nenáročnost na chápání matematických modelů.

Následující kapitola seznamuje čtenáře s jízdními řády a popisuje stávající metody optimalizací turnusů. Dále práce popisuje teorii evolučních algoritmů, kde je důraz kladen na genetické algoritmy, metodu tabu-search a memetické algoritmy. Po teoretickém rozboru je popsán memetický algoritmus optimalizace turnusů jízdních řádů. Popis algoritmu obsahuje definice všech důležitých částí, které jsou potřebné pro jeho implementaci.

Kapitola 2

Jízdní řády

Linkový jízdní řád je komplexní systém zastávek, linek a spojů v nich. Mimo to je jízdní řád tvořen také systémem turnusů, o kterém obyčejný cestující nemá ani ponětí. Pro cestující je jízdním řádem papír, obyčejně přilepený na stěně zastávky, na kterém se může dočíst, kdy přijede jeho dopravní prostředek a kdy se dostane do cíle.

Jízdní řád se v dnešní době musí řídit legislativními nařízeními, která upravují některé jejich aspekty. Jsou tak například upravovány termíny, kdy mohou být jízdní řády měněny. Nařízení také upravují, jak mohou být jízdní řády tvořeny a to ve smyslu aplikovatelných omezení a rozsahu obsluhované oblasti. Dále pak musí být v jízdním řádu zakomponovány vlastnosti obsluhovaného území, musí počítat se stavem vozidel a neméně důležitou měrou musí brát ohled na řidiče, kteří musí mít čas na odpočinek a také mají zákonem stanovenou maximální dobu, po kterou mohou bez přestávky řídit vozidlo[13].

Jízdní řád je obvykle vytvářen na delší časové období, po které je platný. Jízdní řády se tak vytvářejí například na období letních prázdnin. Po období platnosti se jízdní řády nemění, ale mohou obsahovat jisté výjimky. Takovou výjimkou je například vynechání spojů v den státního svátku, protože jsou určeny pro pracující cestující.

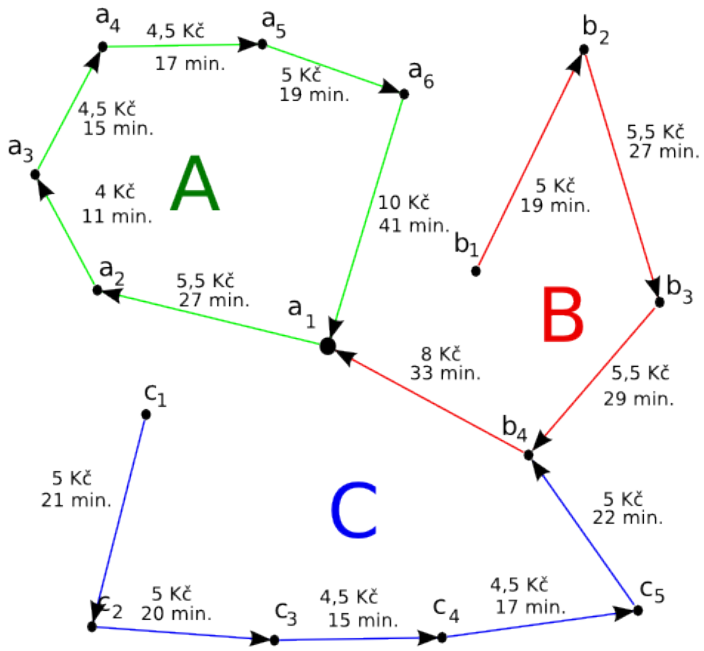
2.1 Matematický popis jízdních řádů

Jízdní řád popisuje systém linek a spojů v dopravní síti. Tak jako každou dopravní síť je možné zapsat pomocí grafu, je možné zapsat grafem i jízdní řád. Toho zápisu je možné využít při optimalizaci jízdních řádů, protože ke grafům je vybudován matematický aparát v podobě teorie grafů.

Každý linkový jízdní řád je tvořen neprázdnou konečnou množinou zastávek, pro které bude jízdní řád vytvářen. Uzlům dopravní sítě tedy odpovídají prvky z množiny zastávek jako vrcholy grafu. Mezi vrcholy jsou ohodnocené orientované hrany. Jejich ohodnocení představuje vzdálenost a ekonomické náklady na přepravu mezi zastávkami a jsou orientovány z výchozí do koncové zastávky. Vytvořením podmnožin množiny zastávek vzniknou linky. Linky jsou uspořádané množiny vrcholů grafu jízdního řádu. Uspořádání linky určuje

cestu grafem jízdního řádů, tedy pořadí obsloužených zastávek. Dále je pak každá linka tvořena množinou spojů obsluhujících zastávky linky v definovaném pořadí. Jeden spoj na jedné lince je posloupnost časových údajů, které označují čas, ve kterém se dopravní prostředek bude nacházet v zástavce[3].

Na obrázku 2.1 je příklad dopravní sítě popsané grafem. V grafu je zeleně vyznačena linka $A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_1\}$, červeně linka $B = \{b_1, b_2, b_3, b_4, a_1\}$ a modře linka $C = \{c_1, c_2, c_3, c_4, c_5, b_4\}$. Každá hrana je ohodnocena časem v minutách a cenou v tisících.



Obrázek 2.1: Ukázka možnosti popisu dopravní sítě grafem s vyznačenými linkami a ohodnocením hran. Uváděné ceny jsou v tisících.

Na dalším obrázku 2.2 jsou tři tabulky představující spoje, které jsou definovány nad linkami z předcházejícího grafu. V prvním řádku každé tabulky je nejdříve uveden identifikátor linky, pro kterou jsou spoje vytvořeny, a dále jsou uvedeny identifikátory spojů. Každý další řádek tabulky obsahuje v prvním sloupci identifikátor zastávky a v dalších čas odjezdu spoje. Rozdíl mezi po sobě jdoucími časy odjezdů v jednom sloupci musí být větší nebo roven času na hraně mezi zastávkami v grafu dopravní sítě. V ukázkových tabulkách byla zvolena dvouminutová proluka modelující nástup a výstup cestujících.

A	1	2	3	4	5	6	7	8	9	10	11	12
a1	06:00	07:00	08:00	09:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00
a2	06:30	07:30	08:30	09:30	10:30	11:30	12:30	13:30	14:30	15:30	16:30	17:30
a3	06:43	07:43	08:43	09:43	10:43	11:43	12:43	13:43	14:43	15:43	16:43	17:43
a4	07:00	08:00	09:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00
a5	07:19	08:19	09:19	10:19	11:19	12:19	13:19	14:19	15:19	16:19	17:19	18:19
a6	07:40	08:40	09:40	10:40	11:40	12:40	13:40	14:40	15:40	16:40	17:40	18:40
a1	08:21	09:21	10:21	11:21	12:21	13:21	14:21	15:21	16:21	17:21	18:21	19:21

B	1	2	3	4	5
b1	05:00	08:00	12:00	15:00	17:00
b2	05:21	08:21	12:21	15:21	17:21
b3	05:50	08:50	12:50	15:50	17:50
b4	06:21	09:21	13:21	16:21	18:21
a1	06:54	09:54	13:54	16:54	18:54

C	1	2	3
c1	06:00	12:00	16:00
c2	06:23	12:23	16:23
c3	06:55	12:55	16:55
c4	07:12	13:12	17:12
c5	07:31	13:31	17:31
b4	07:55	13:55	17:55

Obrázek 2.2: Ukázka vytvořených spojů.

2.2 Turnusy jízdních řádů

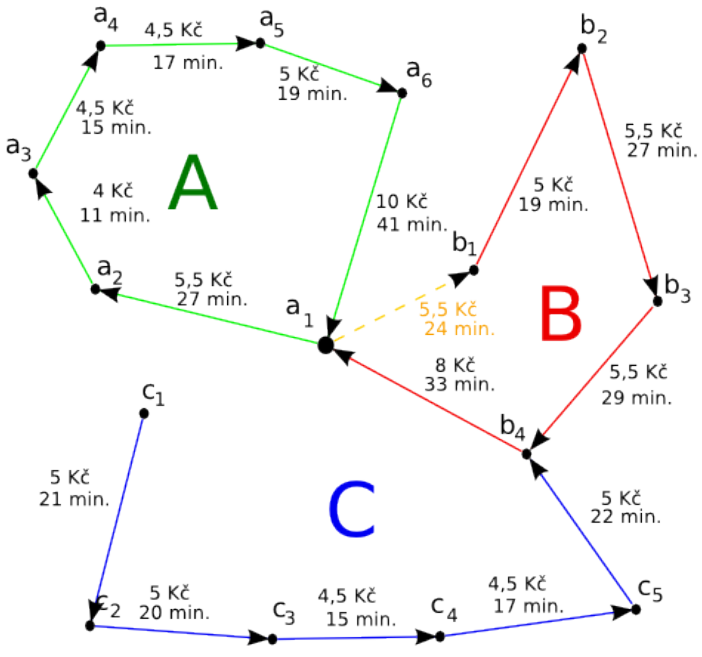
Turnus v jízdním řádu je posloupnost spojů, kterou obsluhuje jeden dopravní prostředek v jednom dni. Každému turnusu je přidělena posádka a těmto posádkám musí být dále sestaveny směny. Turnusy musí být sestaveny tak, aby byly obslouženy všechny spoje v jízdním řádu a aby byl tento úkol splněn co nejekonomičtěji. Ekonomičnost splnění úkolu je dána několika zásadními faktory. Tím prvním je využití co nejmenšího počtu dopravních prostředků. Každý dopravní prostředek navíc znamená náklady na jeho pořízení a jeho údržbu. Se vzrůstajícím počtem dopravních prostředků vzrůstá počet potřebných posádek, který je další důležitým faktorem ekonomičnosti turnusů jízdních řádů. Posádky dále ovlivňují tvorbu turnusů tím, že mají ze zákona určenou maximální dobu řízení vozidla a také musí dodržovat zákonem stanovené přestávky. K turnusům je proto potřeba sestavit i směny posádek. Jeden turnus nemusí odpovídat jedné směně posádky, ty se mohou během turnusu i několikrát vystřídat. Sestavování směn je dalším velmi obtížným úkolem a tato práce se jím dále nebude zabývat.

Každý turnus se sestavuje na celé období platnosti jízdního řádu. Proto tato činnost není příliš náročná na čas. Dopravci znají dostatečně dlouhou dobu dopředu nové jízdní řády a turnusy proto mohou být vytvářeny delší dobu. U tvorby turnusů se také nemusí brát velký ohled na nákladnost tohoto procesu, neboť je vytvářen jen jednou pro celý jízdní řád, který má dlouhou časovou platnost. Toto by však mělo být podmíněno tím, že výsledný systém turnusů bude natolik kvalitní aby dokázal náklady na svou tvorbu během platnosti řádu vrátit.

Turnusy jsou podgrafy orientovaného grafu jízdního řádu. V těchto podgrafech musí být obsaženy všechny hrany grafu jízdního řadů a všechny jeho vrcholy. Navíc mohou obsahovat hrany, které v grafu jízdního řádu nejsou. Tyto hrany navíc znamenají prázdné přejezdy dopravního prostředku mezi stanicemi tak, aby mohl obsloužit jiný spoj. Tak jako v grafu jízdního řádu jsou hrany přejezdů mezi stanicemi ekonomicky a časově ohodnoceny. Při

sestavování turnusů jízdního řádu tvoří taková množina podgrafů, která má nejmenší kardinalitu a nejmenší součet ekonomického ohodnocení na hranách přejezdů mezi stanice za obsluhou jiného spoje, turnusy jízdního řádu. Každý turnus musí obsahovat pouze jednu cestu grafem definující posloupnost obsloužených spojů. V turnusu musí být takové uspořádání spojů, které je možné reálnými dopravními prostředky splnit. Turnus $T = (s_1, s_2, \ldots, s_n)$ je tedy uspořádán n-tice spojů v níž platí uspořádání $s_1 < s_2 < \ldots < s_n$, které znamená, že je možné dostat se z cílové zastávky spoje s_i do výchozí zastávky spoje s_{i+1} pro $i = 1, \ldots, n-1$.

Pro vytváření turnusů není důležité, ve kterých linkách jsou spoje definovány, ale pracuje se sjednocenou množinou všech spojů definovaných ve všech linkách jízdního řádu. Pokud by měly být vytvořeny turnusy pro spoje z obrázku 2.2, bylo by nejdříve nutné vytvořit unikátní jména všech spojů v jízdním řádu. Tohoto lze snadno dosáhnout konkatenací čísla spoje a jména linky, ve které je spoj definován. Pro uváděný příklad by tak vznikla množina dvaceti jmen $\{A1, A2, \ldots, A12, B1, B2, \ldots, B5, C1, C2, C3\}$. Jedním z možných turnusů by byl turnus $(C1, A4, B3, A10)$. V tomto turnusu by dopravní prostředek musel vykonat dva prázdné přejezdy. Prvním prázdný přejezd by byl mezi koncovou zastávkou linky C a počáteční zastávkou linky A. Druhý prázdný přejezd by byl mezi koncovou zastávkou linky A a počáteční zastávkou linky B, který je znázorněn oranžovou orientovanou čerchovanou hranou na obrázku 2.3.



Obrázek 2.3: Turnus zobrazený v grafu sítě tvořený spoji $(C1, A4, B3, A10)$ definovanými v 2.2.

2.3 Definice pojmů

V této části definujeme některé důležité pojmy týkající se turnusů jízdních řádů, které budou později používány.

Důležitým pojmem bude *délka trvání turnusu* , kterou lze spočítat jako rozdíl mezi časem odjezdu z výchozí zastávky prvního obsluhovaného spoje a časem cílové zastávky posledního obsluhovaného spoje. Tato hodnota zahrnuje i čas strávený přejezdy na prázdno.

Dalším pojmem vztahujícím se k času je *aktivní čas turnusu* , který určuje čas strávený dopravním prostředkem obsluhováním spojů v turnusu. Je to tedy délka trvání turnusu bez času stráveného přejezdy na prázdno.

Posledním pojmem je *cena turnusu* vypočítaná jako součet všech cen prázdných přejezdů při plnění turnusu.

2.4 Optimalizace turnusů

V současnosti neexistuje v České republice moderní počítačový systém pro optimalizace turnusů jízdních řádů[2]. V roce 1984 byl vytvořen systém KASTOR, při jehož použití se dosáhlo úspor několik milionů korun.[15] Nicméně na internetu je možné po kratším pátrání nalézt systémy a popis algoritmů založených na nejrůznějších principech. Ovšem žádný není volně dostupný. U komerčních systémů je to pochopitelné, ale ani vědecké články popisující algoritmy není téměř možné získat.

Základním a intuitivním postupem vytvoření turnusů jízdního řádu je manuální uspořádání spojů. Tato metoda je pro většinu firem nejschůdnějším řešením. Postup nezaručuje kvalitu řešení, ale je lehce pochopitelný a snadno proveditelný. Firmy jej volí hlavně kvůli jeho nenáročnosti na prostředky. Na celý postup stačí jedna osoba, případně u složitějších jízdních řádu menší tým lidí. Dále není potřeba chápat složité matematické modely nebo jiné možnosti tvorby.

2.4.1 Optimalizace pomocí celočíselného lineárního programování

Vstupními parametry této metody jsou přepravní požadavky v podobě množin $S = \{1, 2, \ldots, n\}$ spojů z jízdního řádu, potom je n-tice $z = (1, 2, ...n)$ turnus délky n. To znamená, že je-li $(1, 2, \ldots, n)$ turnus, může jeden dopravní prostředek absolvovat spoje $1, 2, \ldots, n$ a přitom splní jízdní řád.

Označme množinu Z jako množinu všech turnusů, které lze s danými přepravními požadavky vytvořit. Dále budeme předpokládat, že na Z je definována nezáporná funkce f, kterou budeme nazývat cenovou funkcí a pro níž platí:

1. f(z)=0 pro každý turnus délky 1
2. pokud je $z = (1, \ldots, k, k+1, \ldots, n), z_1 = (1, \ldots, k), z_2 = (k+1, \ldots, n)$ potom $f(z) = f(z_1) + f(z_2)$

Je zřejmé, že takováto funkce vždy existuje (například $f = 0$). Z definice funkce vyplývá pro $m > 1$ a turnus $z = (1, ..., m)$:

$$f(z) = f((1, ...m)) = \sum_{i=1}^{m-1} f((i, i+1))$$

Další vlastností cenové funkce je $f((n_1, n_2)) = 0$, pokud je cílová stanice spoje n_1 shodná s výchozí stanicí spoje n_2. Jinak se hodnota $f((n_1, n_2))$ rovná nákladům, které jsou nutné pro přesun dopravního prostředku naprázdno z cílové stanice spoje n_1 do výchozí stanice spoje n_2.

Množinu turnusů $Z_0 \subset Z$ nazveme úplnou, pokud pro každé $n \in S$ existuje $z \in Z_0$ takové, že n je složkou turnusu z.

Každá úplná množina turnusů znamená úplné obsloužení jízdního řádu. Přitom počet prvků v Z_0 bude vyjadřovat počet dopravních prostředků, které jsou potřeba pro obsloužení jízdního řádu. $\sum_{z \in Z_0} f(z)$ bude vyjadřovat náklady na přesun dopravního prostředku mezi jednotlivými spoji.

Při sestavování turnusů se poté volí jedna ze tří základních úloh, tak aby se dosáhlo optimální množiny turnusů $Z_0 = \{z_1, \ldots, z_r\}$, která buď:

1. má minimální počet prvků, nebo
2. minimalizuje funkci $c(Z_0) = d * r + \sum_{z \in Z_0} f(z)$ vyjadřující náklady na provoz r dopravních prostředků a na prázdné přesuny v turnusech ze Z_0 (koeficient d se získá z ekonomického rozboru), nebo
3. při pevném $r = r_0$ minimalizuje funkci $\sum_{z \in Z_0} f(z)$

Podrobný popis této metody naleznete v[3]

2.4.2 Sestavení turnusů s pomocí heuristik

Metoda celočíselného programování je ve světě velmi oblíbená a úspěšná. Dočkala se mnohých vylepšení a dalo by se říci, že na ní již není co zlepšovat. Ve světě jízdních řádů se proto pomalu začínají prosazovat zcela jiné přístupy k tomuto problému. Objevují se zatím pouze práce na téma využití různých heuristik při tvorbě časového rozvrhu dopravních prostředků. Jak již bylo mnohokrát zmíněno je tvorba turnusů (časového rozvrhu prostředků) velmi složitá a proto se také nesetkáme s naivními přístupy využívajícími jednoduchých heuristických algoritmů. V drtivé většině případů jsou využívány hybridní genetické algoritmy jako metody hledání optimálních časových rozvrhů. O metodách tvorby turnusů pomocí heuristik se prozatím získávají informace velmi těžko, protože většina děl zabývajících se touto problematikou má komerční charakter a je nutné za ně zaplatit.

Kapitola 3

Evoluční algoritmy

Evoluční algoritmy zahrnují rozsáhlou množinu rozmanitých algoritmů sloužících pro různé účely, ale všechny mají stejný cíl. Tímto společným cílem je nalézt optimální řešení jistého problému. Spojuje je také to, že principy, na kterých tyto algoritmy pracují jsou inspirovány Darwinovou evoluční teorií[6].

Metafora Darwinovy evoluční teorie použitá v evolučních algoritmech obsahuje tyto tři složky[6]:

1. Přirozený výběr, t.j. proces ve kterém jedinci s velkou fitness vstupují do procesu reprodukce s větší pravděpodobností jako jedinci s malou fitness. Pod pojmem fitness rozumíme kvantitativní míru schopnosti přežít a vstupovat do procesu reprodukce.

2. Náhodný genetický drift, ve kterém náhodné události v životě jedince ovlivňují populaci. Takovéto události jsou např. náhodná mutace genetického materiál nebo náhodná smrt jedince s velkou fitness předtím než měl možnost se zúčastnit reprodukčního procesu. Náhodné efekty genetického driftu jsou významné hlavně pro malé populace.

3. Reprodukční proces, v rámci kterého se z rodičů vytvářejí potomci. Genetické informace potomků se vytváří vzájemnou výměnou genetické informace rodičů. Nejčastěji tento proces probíhá tak, že z genetické informace dvou jedinců se náhodně vyberou části chromozómu, ze kterých se potom sestaví genetické informace nového jedince - potomka. Tento proces se jinak nazývá křížení a vzhledem k tomu, že se vyskytuje u všech organizmů můžeme usoudit, že podstatně zvyšuje rychlost a účinnost evoluce.

Základním pojmem evolučních algoritmů je chromozóm. Jde o zakódování problému do podoby srozumitelné počítači. V každém chromozómu je zakódováno právě jedno řešení úlohy. Chromozómy, které obsahují kvalitní řešení mají také velkou hodnotu fitness funkce a tím pádem se také s větší pravděpodobností účastní reprodukčního procesu. Reprodukce neobsahuje pouze křížení chromozómů, ale zahrnuje také mutaci. Mutace umožňuje vytvořit nové kombinace, které se v populaci ještě nevyskytují, a tím rozšířit prohledávaný prostor.

Po určitém počtu generací se v populaci začnou vyskytovat vysoce ohodnocené chromozómy, které reprezentují velmi kvalitní řešení optimalizačního problému[6].

Evoluční algoritmy představují, díky své schopností řešit velmi složité problémy, netradiční přístup hledání optimálního řešení. Jejich hlavní předností je schopnost prohledávat velmi rozsáhlé prostory možných řešení a nalézt nové a neočekávané řešení problému.

3.1 Horolezecký algoritmus

Patří k těm nejjednodušším algoritmům. V angličtině se jmenuje Hill-Climbing a v jeho průběhu je skutečně možné pozorovat něco jako šplhání po horách. Jeho popis bude ve stručnosti uveden, protože některé další metody z něj vycházejí. Kromě toho, že se jedná o stochastický proces nemá s evolučními algoritmy téměř nic společného. Jeho činnost spočívá v náhodném prohledávání okolí jednoho zvoleného řešení. Na počátku je toto zvolené řešení náhodně vygenerováno. V průběhu algoritmu jsou generována řešení v blízkosti vybraného řešení. Z okolí zvoleného řešení se vybere to nejlepší jako další zvolené a opět se generuje jeho okolí. Algoritmus průběžně zaznamenává nejlepší nalezené řešení, které je po zvoleném počtu iterací vráceno jako výsledek[6].

Následující pseudokód stručně vystihuje podstatu horolezeckého algoritmu.

Algoritmus 1 : Horelezecký algoritmus[6].

input: t_{max}; output: f_{fin}, α_{fin}
$\alpha :=$ náhodně vygenerované řešení
$f_{fin} := \infty; t := 0;$
while $t < t_{max}$ **do**
  $t := t + 1;$
  $\alpha^* :=$ nejlepší jedinec z okolí
  if $f(\alpha^*) < f_{fin}$ **then**
    $f_{fin} := f(\alpha^*)$
    $\alpha_{fin} := \alpha^*$
  end if
  $\alpha := \alpha^*$
end while

Jedná se o velmi jednoduchý algoritmus, který dobře funguje na problémy, jež nemají mnoho lokálních extrémů. Každý takový lokální extrém totiž může způsobit uváznutí algoritmu a tím pádem nenalezení globálního extrému. Problém uváznutí je možné řešit nastavením větší vzdálenosti generovaného okolí zvoleného řešení. Takovýto postup umožní algoritmu překonat lokální extrém, ale na druhou stranu může znemožnit nalezení globálního extrému.

Tvorbu okolí zvoleného jedince lze přirovnat k mutaci při evoluci, díky čemu je možné o algoritmu mluvit jako o evolučním.

3.2 Zakázané hledání

Metoda s anglickým názvem tabu search je založena na horolezeckém algoritmu. Tato metoda se snaží omezit problém návratu horolezeckého algoritmu do předchozího řešení pomocí jednoduchého vylepšení. Uváznutí horolezeckého algoritmu je způsobeno postupným návratem z okolí do předchozího nejlepšího řešení. Kdyby si algoritmus určitým způsobem pamatoval cestu, po které již šel, k tomuto uváznutí by nedocházelo. Metoda zakázaného hledání používá jako paměť cesty množinu zakázaných operací pro generování okolí zvoleného řešení.

Algoritmus 2 : Zakázané hledání[6].

input: t_{max}, s; output: f_{fin}, α_{fin}
$\alpha :=$ náhodně vygenerované řešení
$f_{fin} := \infty; t := 0; T := \emptyset;$
while $t < t_{max}$ **do**
  $t := t + 1; f_{fin-loc} := \infty;$
  $\alpha^* := t\alpha$
  if $t \ni T and f(\alpha^*) < f_{fin-loc} or f(\alpha^*) < f_{fin}$ **then**
    $f_{fin-loc} := f(\alpha^*);$
    $\alpha' := \alpha^*;$
    $t^* := t;$
  end if
  if $f_{fin-loc} < f_{fin}$ **then**
    $f_{fin} := f_{fin-loc};$
    $\alpha_{fin} := \alpha';$
  end if
  $\alpha := \alpha';$
  if $|T| < s$ **then**
    $T := T \cup \{t^{*-1}\}$
  else
    $T := (T \cup \{t^{*-1}\}) - \{t^s\}$
  end if
end while

Její princip je dobře popsatelný na hledání řešení s binárním řetězcem. Metoda bude vytvářet okolí řešení pomocí změny jednoho bitu v aktuálním zvoleném řešení. Po vytvoření okolí je vybráno to nejlepší řešení z nich. Horolezecký algoritmus může v příštím vygenerovaném okolí vytvořit opět předcházející řešení, které bude nejkvalitnější, a vrátí se zpět do něj. Metoda zakázaného hledání se tomuto snaží zabránit pomocí množiny zakázaných operací, které znemožní rychlý návrat algoritmu do předchozího řešení. V případě binárního řetězce se do množiny zakázaných transformací ukládají indexy bitů, které se pro genero-

vání příštího okolí nesmí používat. Tento postup tak umožní algoritmu překonávat lokální extrémy. Množina zakázaných operací musí být konečná, jinak by po čase nebylo možné vytvářet další okolí, protože by byly všechny transformace zakázány. Díky tomuto omezení není zcela zaručeno, že se metoda nikdy nevrátí do lokálního extrému. Čím je množina zakázaných operací větší tím je menší pravděpodobnost návratu zpět, ale na druhou stranu je větší pravděpodobnost znemožnění nalezení globálního extrému.

Princip metody zakázaného hledání je popsán v pseudokódu 2.

3.3 Genetické algoritmy

Genetické algoritmy patří mezi základní stochastické optimalizační algoritmy s výraznými evolučními rysy. V současnosti jsou nejčastěji používaným evolučním optimalizačním algoritmem, se širokou paletou aplikací od optimalizací multimodálních funkcí přes kombinatorické a grafově-teoretické problémy až po aplikace nazývané umělý život[6].

Předchozí algoritmy jsou jednoduché a jejich omezení spočívá hlavně v tom, že slepě generují řešení bez vztahu k optimalizované funkci. Numerické metody řeší tento problém tak, že buď využívají poznatků o tvaru funkce v daném bodě, a nebo jsou schopné pomocí předcházejících řešení indikovat s velkou efektivností směr minimalizace funkce. Ukazuje se, že na základě analogie s evolučními procesy probíhajícím v biologických systémech existuje alternativní možnost, jak usměrnit náhodné generování bodů směrem k optimálním hodnotám. Právě tato analogie se stala základem genetického algoritmu, který vylepšuje čistě stochastické slepé algoritmy tak, že poskytuje v reálném čase optimální řešení[6].

Darwinova teorie evoluce se zakládá na tezi přirozeného výběru, podle kterého přežívají jen ti nejlépe přizpůsobení jedinci populace. Reprodukcí dvou jedinců s vysokou fitness získáme s vysokou pravděpodobností jedince, kteří budou dobře přizpůsobení pro přežití. Při podrobné analýze se ukazuje, že samostatné působení reprodukce není dostatečně efektivní pro umožnění vzniku jedinců s novými vlastnostmi, které zvyšují schopnost přežití jedinců. Do evoluce v přírodě se zapojuje mutace, která náhodným způsobem ovlivňuje genetický materiál populace[6].

Pojem fitness, který hraje klíčovou roli v úvahách o genetických algoritmech, je v biologii definován jako relativní schopnost jedince přežít a reprodukovat se v daném prostředí. Stejně tak je chápána i v genetických algoritmech. Je to číslo přiřazené informaci zakódované v chromozómu reprezentující řešení problému[6].

Na vysokém stupni abstrakce lze biologického jedince nahradit pojmem chromozóm, který reprezentuje lineárně uspořádaným způsobem informační obsah jedince. Potom lze mluvit o populaci chromozomů, které se účastní reprodukce s pravděpodobností úměrnou jejich fitness, přičemž integrální součástí této reprodukce je mutace. Jedinci představovaní chromozomy jsou lineární řetězce symbolů a populace P je množina těchto chromozómů:

$$P = \{\alpha_1, \alpha_2, \ldots, \alpha_p\} \subseteq \{a, b, \ldots\}^k$$

Populace P obsahuje p chromozómů, které jsou realizovány jako řetězce délky k ze znaků $a, b, \ldots$. Každý chromozóm je ohodnocený fitness, která se interpretuje jako zobrazení chromozómu na reálné číslo

$$F\colon P \to R_+$$

Proces reprodukce chromozómů začíná tím, že se kvazináhodně vyberou dva chromozómy se závislostí na jejich fitness. Pod reprodukcí rozumíme takový proces, ve kterém se dva původní chromozómy α_1 a α_2 reprodukují na dva chromozómy α_1' a α_2'

$$(\alpha_1', \alpha_2') = O_{repro}(\alpha_1, \alpha_2)$$

Tento operátor obsahuje část křížení a část mutace, přičemž obě dvě části se provádějí jen s určitou pravděpodobnosti[6]. Pokud při reprodukci vzniknou nové chromozómy je nutné je zařadit do stávající generace chromozómů, tak aby se mohly účastnit evoluce.

3.3.1 Selekce

Do reprodukčního procesu musí být vybírány chromozómy s jistou mírou náhodnosti. Pokud by byly vybírány jen ty s největší hodnotou fitness, potom by s určitou pravděpodobností zůstalo hledaní optimálního řešení jen v okolí nejsilnějšího jedince, který by se objevil v průběhu algoritmu. Tento silný jedinec by však nemusel být globálním extrémem a algoritmus by tak na úkor dočasně lepšího výsledku uvázl v lokálním extrému.

Nejběžnějším způsobem výběru jednotlivých chromozómů je pomocí *roulet wheel* algoritmu. Jde v podstatě o takové kolo štěstí. Toto kolo štěstí je rozděleno na tolik dílů kolik je jedinců v populaci. Každý díl je přiřazen jednomu jedinci a velikost jeho dílu je závislá na jeho fitness. Algoritmus může díky závislosti pravděpodobnosti na jeho fitness uváznout v lokálním extrému a proto se hodnoty pravděpodobnosti jistým způsobem normalizují tak, aby se smyl rozdíl mezi nejhorším a nejlepším jedincem jedincem v populaci[9].

Jedním z dalších algoritmů pro výběr jedinců z populace je turnajový algoritmus. Při výběru jsou náhodně zvoleni dva jedinci z populace a ti se spolu utkávají v turnaji pomocí porovnání svých fitness. V turnaji zvítězí ten jedinec, který má větší fitness, a je vybrán pro účast v reprodukčním procesu[9].

3.3.2 Křížení

Křížení znamená vytváření nových potomků pomocí existujících jedinců (rodičů). Nový jedinec (potomek) může mít větší fitness než jeho rodiče, ale také může mít fitness menší. Pro genetické algoritmy je vytvořeno mnoho typů operátorů křížení. Použitý operátor křížení v genetickém algoritmu závisí na typu řešeného problému a zakódování řešení do chromozómu.

Nejběžnějším operátorem křížení je jednobodové nebo dvoubodové křížení. Jejich principem je náhodné stanovení indexu v chromozómu, který určuje bod křížení. Po zvolení bodu křížení dojde k tvorbě potomků pomocí spojení a záměny jednotlivých částí rodičů

určených bodem křížení. Dalším operátorem křížení je uniformní křížení, které vytváří potomky z rodičovských chromozómů pomocí náhodně vygenerované binární masky. Maska tedy obsahuje pouze hodnoty 1 a 0 a je stejně dlouhá jako rodičovské chromozómy. Při vytváření potomka se vytvořená maska prochází znak po znaku a podle aktuální hodnoty znaku se GEN kopíruje z prvního nebo z druhého rodiče. Uniformní operátor vnáší do populace velké množství nové informace a je vhodný pro problémy obsahující velké množství lokálních extrémů způsobujících předčasnou konvergenci algoritmu[9].

Pro chromozómy skládající se z permutace prvků jsou vytvořeny speciální operátory křížení, protože jinak dochází k porušení permutací uvnitř chromozómu a tyto permutace se poté musí složitě opravovat. Základním operátorem křížení pro permutační chromozómy je operátor *Order 1 crossover* (OX)[9]. Jeho aplikací vzniká pouze jeden potomek. Sestává se ze tří kroků. V prvním kroku náhodně vybere dva body křížení. V druhém kroku zkopíruje hodnoty genů mezi body křížení z prvního rodiče. A v posledním třetím kroku prochází obsah druhého rodiče od místa druhého bodu křížení a kopíruje jeho hodnoty genů do potomka, ale pouze pokud je potomek ještě neobsahuje. Tento princip rozšiřuje operátor křížení pojmenovaný *Partial mapped crossover* (PMX)[9].

PMX se sestává také ze tří kroků. V prvním kroku se stejně jako u OX zvolí dva náhodné body křížení a zkopírují se hodnoty genů mezi body křížení z prvního rodiče do potomka. V druhém kroku se prochází postupně geny mezi body křížení v prvním rodiči. Aktuální gen v prvním rodiči se spojí s genem na stejném místě v druhém rodiči a s genem se stejnou hodnotou. Pokud protilehlý gen nemá stejnou hodnotu, jsou hodnoty genů v druhém rodiči mezi sebou zaměněny. Po zkontrolování všech genů mezi body křížení v prvním rodiči jsou přeneseny do potomka ty geny z druhého rodiče, u kterých byla provedena záměna a nenacházejí se mezi body křížení, přičemž zachovává pozice těchto genů. V posledním třetím kroku postupně do potomka kopíruje geny z druhého rodiče počínaje genem nacházejícím se za druhým bodem křížení. Jsou kopírovány pouze ty geny, které potomek ještě neobsahuje.

3.3.3 Mutace

V evolučních algoritmech stejně jako v evoluční teorii hraje životně důležitou úlohu. Je to proces, při kterém dochází k náhodné změně některých vlastností - parametrů daného jedince. Jednoduše řečeno dochází k mutaci. Tento operátor má velmi malou četnost výskytu, protože velmi zásadně ovlivňuje populaci. Příliš velká pravděpodobnost mutace může způsobit nestabilitu algoritmu a naopak příliš malá mutace může znemožnit konvergenci do globálního minima. Operátor mutace pro binárně kódované chromozómy má typicky tvar inverze jednoho z genů. Pro permutačně kódované chromozómy mívá tvar záměny hodnot dvou genů[9].

3.4 Memetické algoritmy

Memetické algoritmy představují širokou třídu metahueristických algoritmů. Klíčovou charakteristikou těchto algoritmů je použití technik lokálního vyhledáváni, speciálních rekombinačních operátorů apod[14]. Některé vědecké práce označují memetické algoritmy jako hybridní genetické algoritmy. Jsou vhodné k řešení složitých optimalizačních problému s rozsáhlými chromozómy[12]. Činnosti memetického algoritmu je možné popsat následujícím pseudokódem:

Algoritmus 3 : Schéma činnosti memetického algoritmu.

```
Inicializace populace
Ohodnocení jedinců v populaci
while dokud není splněna podmínka ukončení do
    Selekce jedinců pro reprodukci
    Reprodukce jedinců křížením a mutací
    Aplikace lokálního vyhledávání
    Obnova populace pomocí nových jedinců
end while
```

Memetika je mladý vědní obor zabývající se přenosem informací mezi rodiči a potomky negenetickou cestou. Každá takováto přenášená informace je nazývána mem a přenáší se díky procesu, který lze nazvat napodobování. Ačkoli se memetická algoritmy poprvé objevily již v roce 1989 jsou momentálně předmět aktivního výzkumu.

Kapitola 4

Návrh algoritmu optimalizace

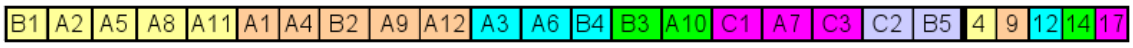
Sestavení optimálních turnusů jízdních řádů je velmi složitý problém a algoritmus pro jeho řešení zřejmě nebude zcela běžným evolučním algoritmem. Pro jeho řešení bude třeba mnoho dodatečných zdrojů informací, jako je cenová tabulka přejezdů mezi stanicemi nebo maximální možná délka práce řidiče. Navíc musí být algoritmus snadno modifikovatelný, protože firmy mohou mít různé požadavky na vlastnosti výsledných turnusů a také se mohou měnit omezující podmínky dané zákony.

Algoritmus bude pracovat ve více krocích, protože není možné vytvořit turnusy pro všechny spoje v celé délce platnosti jízdního řádu v jednom kroku. Prvním důvodem proč bude algoritmus obsahovat více kroků je ten, že turnusy se sestavují pro jeden den. V jízdních řádech se dá vysledovat perioda opakování s délkou jednoho týdne. V ideálním případě by stačilo pro vytvoření všech turnusů jízdního řádu sedm kroků, kdy by se pro každý den v týdnu vytvořily turnusy na základě spojů aktivních ten den. Bohužel v jízdních řádech se vyskytuje mnoho omezení upravující provoz spojů v průběhu platnosti jízdního řádu. Algoritmus tedy bude vytvářet turnusy den po dni platnosti jízdního řádu. Je zaručeno jeho ukončení, protože jízdní řády mají omezenou dobu platnosti. V každém kroku se tedy vytvoří množina spojů, které jsou v tom dni provozovány a pro tuto množinu se vytvoří turnusy. Aby se předešlo zbytečnému počítání, budou množiny spojů provozovaných v jednom dni porovnávány s množinami, pro které již turnusy existují, a pokud se budou shodovat, bude použito již vytvořených turnusů. Kdyby nebylo využíváno turnusů již vytvořených pro stejné množiny, algoritmus by byl velmi neefektivní a hloupý, protože v jízdních řádech se množiny spojů provozovaných v jednom dni vyskytují v mnohem větším počtu než ty množiny, které jsou pro jízdní řád unikátní a znemožňují vytvoření turnusů v jednom kroku. Díky využívání vytvořených turnusů algoritmus s velkou pravděpodobností neprovede víc jak 20 kroků. Výpočet algoritmu bude probíhat tak jak bylo popsán v algoritmu 3. Pro lokální vyhledávání bude použít algoritmus tabu-search, který se bude snažit nalézt optimální rozdělení chromozómu do turnusů. Podrobnosti o chromozómu, operátorech křížení a mutace, a hledání optimálního rozdělení chromozómu je uvedeno v následujících kapitolách.

4.1 Zakódování problému

Řešený optimalizační problém není běžný a ani zakódování problému nebude zcela běžné. Jelikož turnusy musí obsahovat všechny spoje jízdního řádu bude použito permutační kódování chromozómu. Celý chromozóm se navíc bude skládat ze dvou částí. První část chromozómu, s pevnou délkou, bude obsahovat permutaci čísel spojů jednoznačně identifikujících spoj v jízdním řádu. Z permutace v první části se budou vybírat úseky, tvořící již konkrétní turnusy. Turnusy ve vybraných úsecích první části budou dány uspořádáním čísel spojů a dále s nimi nebude manipulováno. Vybrané úseky z první části budou uloženy v druhé části chromozómu v podobě indexů určujících hranice úseků z první části. Druhá část chromozómu bude tedy mít proměnlivou délku a hodnoty jejích genů budou indexy v první části chromozómu, udávající poslední spoj v turnusu. Pro konec posledního turnusu bude hranice dána koncem první části. Pokud tedy bude celá permutace z první části určovat jeden turnus bude druhá část prázdná. Je jisté, že v takovémto chromozómu je možné vždy nalézt nejméně jednu množinu turnusů, pokrývající celý jízdní řád, takovou že druhá část bude obsahovat všechny indexy spojů z první části vyjma posledního. Taková množina turnusů znamená, že každý spoj je obsluhován jedním dopravním prostředkem.

Na obrázku 4.1 je vytvořen chromozóm pro spoje z obrázku 2.2. V první části chromozómu se nachází permutace všech spojů jízdního řádu. Jména spojů jsou konkatenována se jmény linek, do kterých spoje patří. Barevně jsou označeny skupiny spojů tvořící turnusy. Barvy skupin spojů v první části odpovídají barvám v části druhé. Chromozóm tedy představuje turnusy $(B1, A2, A5, A8, A11)$, $(A1, A4, B2, A9, A12)$, $(A3, A6, B4)$, $(B3, A10)$, $(C1, A7, C3)$ a $(C2, B5)$.



Obrázek 4.1: Ilustrace chromozómu.

4.2 Fitness funkce

Fitness funkce bude v praxi hodnotit velké množství proměnlivých atributů vytvořených turnusů. Jejím základním cílem bude ohodnotit ekonomičnost vytvořených turnusů. Pro účely této práce postačí fitness funkce taková, která bude dostatečně upřednostňovat řešení s menším počtem turnusů s minimálními náklady na prázdné přejezdy dopravních prostředků mezi koncovými a výchozími zastávkami spojů v turnusu. Tedy bude hledat takovou množinu turnusů, kterou bude moci uskutečnit s minimálními náklady[4].

V tomto případě tedy půjde o funkci $f(Z_0) = c_p * k + \sum_{z \in Z_0} c(z)$, která byla již popsána v matematické metodě sestavování turnusů. Koeficient c_p bude představovat náklady na provoz jednoho dopravního prostředku. Hodnota proměnné k bude představovat počet vytvořených turnusů. A $\sum_{z \in Z_0} c(z)$ bude znamenat cenu přejezdů dopravního prostředku

na prázdno. S takto definovanou fitness funkcí se bude snažit algoritmus minimalizovat provozní náklady.

4.3 Operátor křížení

Pro permutační kódování chromozómů jsou vytvořeny speciální operátory zachovávající permutaci v platném stavu. Pro případ turnusů bude využit operátor PMX v mírně upravené verzi, která se bude snažit nerozbíjet kvalitní části chromozómů, a bude aplikován pouze na první část chromozómu. V základní podobě PMX vytváří náhodné body křížení a s nimi dále pracuje. Tento postup nebere ohled na kvalitní části a může je zničit. Zničení kvalitních částí není problém pokud mají chromozómy rozumnou délku a algoritmus má dostatek času se k nim vrátit. V případě turnusů jízdních řádů mohou chromozómy obsahovat až tisíce spojů, pro které budou turnusy tvořeny. Operátor křížení se tedy bude snažit kvalitní části chromozómu nalézt, zachovat a šířit do dalších generací.

Princip operátoru PMX zůstane zachován až na vytváření bodů křížení. Ty budou nově vytvářeny v místech kde začínají a končí kvalitní úseky chromozómu. Díky takto vytvářeným novým jedincům se budou v populaci šířit kvalitní turnusy dle hypotézy o stavebních blocích[6]. Ta tvrdí, že krátká a nadprůměrně ohodnocená schémata se v populaci rychle šíří. Takováto schémata jsou nazývána stavebními bloky a právě z nich se skládá chromozóm. V našem případě se chromozóm bude skládat z nadprůměrně ohodnocených turnusů.

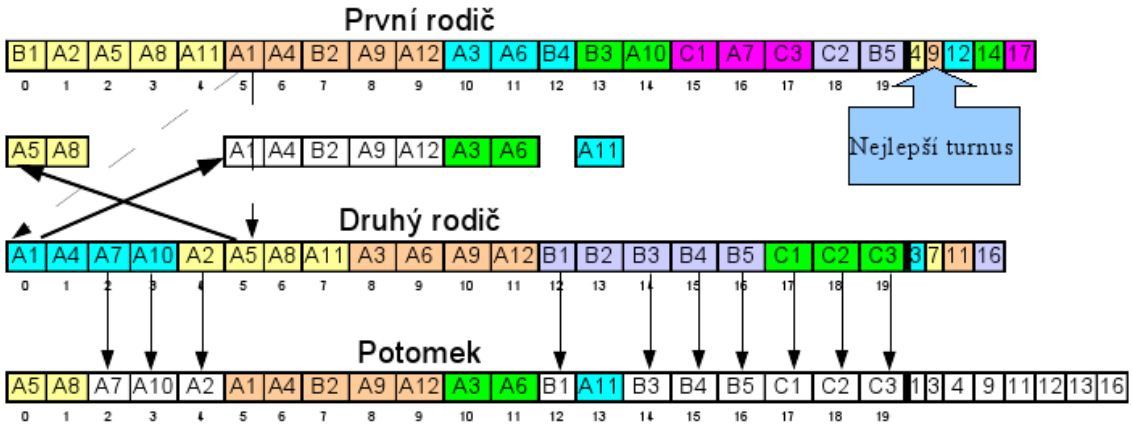
V běžné praxi není třeba stavební bloky vyhledávat a upřednostňovat je tímto způsobem, spíše to povede k předčasné konvergenci algoritmu do lokálního extrému. Náš případ je mnohem složitější. První část chromozómu, nad kterou je aplikován operátor křížení, neví vůbec nic o tom jaké turnusy jsou z ní vytvořeny, to je definováno až v druhé části. Díky tomuto tvaru chromozómu by bez speciálního operátoru křížení docházelo k velkému rozvracení stavebních bloků a konvergence algoritmu by znatelně zpomalila. Hlavním problémem je, že turnusy z permutace v první části se tvoří v druhé části pomocí speciálního algoritmu, který po křížení nemusí nalézt původní turnus. Nenalezení původního turnusu by jen znamenalo, že nebyl dostatečně kvalitní a byl nahrazen lepším řešením. S vysokou pravděpodobností původní turnus nalezen bude, a jeho hledání bude zbytečně zdržovat průběh algoritmu.

Do dalších generací algoritmu budou kopírovány úseky z první části chromozómu a v druhé části k nim budou vytvářeny příslušné hranice. Výběr kopírovaných úseků, tedy bodů křížení, bude prováděn na základě lokální fitness funkce pro tyto úseky (turnusy). Tato lokální fitness funkce bude upřednostňovat turnusy, které obsahují málo přejezdů mezi zastávkami a jsou tak ekonomicky výhodnější. Dalším atributem, podle kterého budou turnusy vybírány bude poměr mezi aktivním časem turnusu a maximální délkou turnusu. Lokální fitness funkce bude mít tvar:

$$f_l(t) = (1 - \frac{c(t)}{\sum_{z \in Z_0} c(z)}) + \frac{d_a(t)}{d_{max}}$$

Kde poměr $\frac{c(t)}{\sum_{z \in Z_0} c(z)}$ je normalizovaná cena turnusu. $\frac{d_a(t)}{d_{max}}$ je poměr délky trvání turnusu a maximální možné délky. Tento poměr je ve funkci důležitý, protože umožňuje upřednostňovat turnusy, které obsahují více jak jeden spoj. Bez tohoto poměru by byly turnusy vybírány pouze na základě jejich ceny a tím pádem by téměř vždy zvítězil turnus jednotkové délky (cenu turnusu z velké části tvoří náklady na pohonné hmoty, které se v krátkých turnusech spotřebuje méně).

Průběh křížení chromozómů obsahující permutace spojů z obrázku 2.2 je názorně zobrazen na obrázku 4.2.

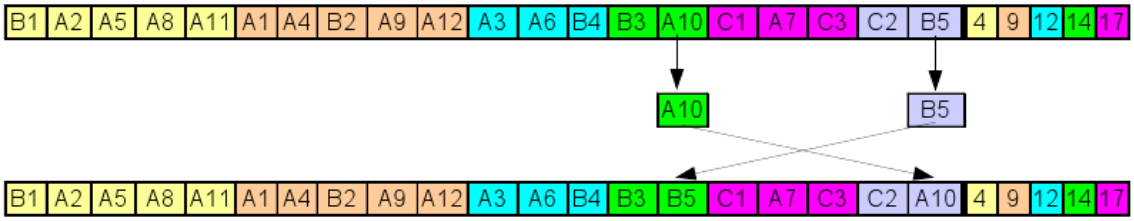


Obrázek 4.2: Ilustrace křížení.

4.4 Operátor mutace

Mutace bude vnášet do nově vytvářených jedinců novou informaci a tím zvětšovat diverzitu populace. Díky speciálnímu operátoru křížení, který se snaží vyhledávat stavební bloky a přenášet je do dalších generací může nastat problém s uváznutím v lokálním extrému. Z tohoto důvodu bude hrát mutace důležitou roli. Bude aplikována s větší pravděpodobností než je tomu ve standardních genetických algoritmech. Hodnotu této pravděpodobnosti nelze předem určit.

Existuje několik implementací operátoru mutace, ale většinu není možné použít v permutačních chromozómech. V chromozómech tvořených binárním řetězcem je možné použít například inverzi bitu[6], ale v permutaci toto není možné. Vhledem ke specifickým potřebám permutačního kódování chromozómu bude operátorem mutace záměna hodnot dvou genů uvnitř chromozómu (Obrázek 4.3). Po provedení mutace tímto způsobem zůstane permutace uvnitř chromozómu neporušená.

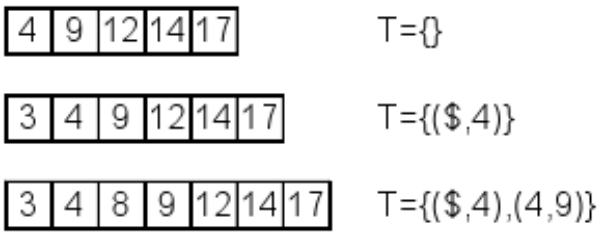


Obrázek 4.3: Ilustrace mutace.

4.5 Algoritmus vytváření turnusů

V první části chromozómu je uložena permutace spojů jízdního řádu. Tato permutace je rozdělena do několika částí, čímž jsou vlastně vytvořeny turnusy jízdního řádu. V první části chromozómu není možné označit úseky jako turnusy a proto obsahuje druhou část udávající hranice jednotlivých turnusů uvnitř permutace z první části. Hledáním hranic určující turnusy se bude zabývat algoritmus v podobě tabu search.

V inicializaci algoritmu bude vytvořeno první řešení pomocí průchodu permutací s hledáním porušení uspořádání nad spoji $s_i < s_{i+1}$. Narazí-li algoritmus v průběhu průchodu permutací na dvojici turnusů $s_i > s_{i+1}$ potom bude index i přidán na konec druhé části. Druhou možností jak ohraničit jeden turnus je překročení maximální délky trvání turnusu, zahrnující i nemožnost splnění přestávkové povinnosti řidiče. Do druhé části se tak přidá index spoje před spojem, jehož přidání do turnusu poruší podmínku maximální délky trvání.



Obrázek 4.4: Ilustrace lokálního vyhledávání hranic turnusů pomocí množiny zakázaných operací.

V dalších iteracích bude algoritmus pomocí několika operací vytvářet nové hranice. Tyto operace budou prováděny tak, aby vlastnosti nových turnusů neporušovaly podmínky platného turnusu. V podstatě půjde pouze o jednu operaci posunující hranice turnusu pomocí inkrementace nebo dekrementace hraničního indexu. Po takovéto změně hranic budou zkontrolovány vlastnosti následujících turnusů a případně upraveny jejich hranice. Tabu-search pracuje s množinou zakázaných operací, kterou bude v této implementaci tvořit seznam změněných hranic turnusů. Algoritmus tedy bude zakazovat operace, které povedou k vytvoření původních turnusů.

Na obrázku 4.4 je znázorněn možný vývoj hranic turnusů pro případ chromozómu z obrázku 4.2. Algoritmus v množině zakázaných operací uchovává hranice turnusu, který byl změněn, a nedovolí jeho brzké obnovení. Znak $ je zástupná hodnota pro konec chromozómu. Může se vyskytovat jak na první tak na druhé pozici.

Kapitola 5

Analýza požadavků na optimalizační systém

V této kapitole se zaměříme na rozbor požadavků programového vybavení vytvářeného systému pro optimalizaci turnusů jízdních řádů. V první části analýzy budou podrobně rozebrány požadavky na ovládání systému. Druhá část analýzy se zaměří na identifikaci částí systému, které bude potřeba často upravovat.

5.1 Požadavky na ovládání

Ovládání systému musí být intuitivní a odolné vůči chybám uživatelů. Je možné předpokládat, že proces tvorby optimálních turnusů bude probíhat poměrně dlouhou dobu. Kdyby při spuštění udělal uživatel nějakou chybu, například špatně nastavil parametry optimalizace, tak čas strávený výpočtem by nebyl využit užitečně a optimalizace by se tím mohla prodražit. Je proto žádoucí, aby systém poskytoval uživatelům komplexní přehled o průběhu výpočtu. Díky informacím o průběhu optimalizace budou moci uživatelé znalí problematiky tvorby turnusů posoudit, zda se optimalizace vyvíjí správným směrem. Pokud shledají průběžné výsledky jako nedostatečné, budou moci upravit parametry optimalizace dříve než po získání celkového výsledku, který by nemusel být dostatečně kvalitní.

Jak již bylo zmíněno, systém musí poskytovat možnost změny parametrů optimalizace. Protože parametrů optimalizace bude více, je vhodné umožnit uživatelům zapsat je do konfiguračního souboru. Pro možnost rychlé změny by mělo být možné upravit parametry pomocí argumentů příkazové řádky. V konfiguračním souboru budou názvy všech parametrů optimalizace i s jejich hodnotami. Na příkazové řádce budou uživatelé moci nastavit libovolný z parametrů uvedený v konfiguračním souboru. Hodnoty z konfiguračního souboru budou použity jako výchozí, pokud nebude parametr nastaven pomocí příkazové řádky.

Protože bude možné ovlivnit parametry optimalizace ze dvou míst (konfiguračního souboru a příkazové řádky), je žádoucí, aby systém před zahájením optimalizace všechny parametry zobrazil a dal uživateli na výběr, zda chce spustit výpočet s těmito konkrétními

parametry. Toto opatření se opět snaží zabránit běhům algoritmu, které mají velmi malou pravděpodobnost na získaní vhodných výsledků.

5.1.1 Identifikace parametrů

Jelikož je k optimalizaci turnusů používán memetický algoritmus, což je v podstatě upravený genetický algoritmus, bude systém zpracovávat parametry obvyklé u genetických algoritmů. Uživatelé tak budou moci nastavovat velikost populace, pravděpodobnost křížení, pravděpodobnost mutace, velikost nahrazované populace a počet generací algoritmu.

Další parametry se budou vztahovat k úpravě genetického algoritmu. Prvním je hranice, při které je turnus považován za vhodný. Tato hranice se bude využívat při křížení jedinců. V návrhu algoritmu byl popsán mechanismus křížení s využitím bodů křížení určených vhodnými turnusy. Právě hodnota tohoto parametru bude určovat, zda je turnus dostatečně vhodný a zda se jeho krajní body použijí jako body křížení. Nakonec tu jsou dva parametry vztahující se k algoritmu tabu search. Je to počet cyklů algoritmu a velikost množiny zakázaných operací.

Pro správnou funkci bude systém potřebovat několik dalších parametrů. Některé z nich se ještě týkají algoritmu optimalizace, ale především patří k turnusům. Jedná se o parametry popisující legislativní omezení a požadavky kladené na vlastnosti tvořených turnusů. V parametrech systému tedy budou časy povinných přestávek i maximální délka turnusu. Délka přestávek je povinná a nelze ji žádným způsobem měnit. Maximální délka turnusu je legislativního nařízení omezující maximální možnou dobu délky řízení řidiče. V aktuálním znění je maximální povolená doba řízení 10 hodin. Dle zákona můžou řidiči řídit po maximální dobu pouze dvakrát do týdne, pokud musí řídit každý den nesmí délka přesáhnout 9 hodin. Proto bude systém přijímat parametr nejlepší délky turnusu. Tento parametr bude odpovídat době řízení, kterou mohou řidič vykonávat každý den. Omezení na délku turnusu bude uplatněno pouze tehdy, když nebude žádoucí, aby se řidiči střídali během jednoho turnusu.

Další parametry ovliňující vlastnosti turnusů jsou pouze ekonomického charakteru. Pro výpočet vhodnosti jednoho turnusu je potřeba znát cenu čekání, cenu prázdné jízdy, cenu za turnus delší než nejvhodnější délka a cenu za turnus kratší než nejvhodnější délka. Poslední dva uvedené parametry se uplatní pouze opět, nebude-li žádoucí tvořit turnusy maximální délky. Posledními ekonomickými parametry a parametry vztahujícími se k turnusům jsou cena jednotky vzdálenosti, cena jednoho dopravního prostředku a cena jednotky čekání. Tyto parametry budou použity pro výpočet celkové ceny turnusu a celého systému turnusů. Cena čekání se v parametrech vyskytuje dvakrát. Poprvé u výpočtu vhodnosti jednoho turnusu a podruhé u výpočtu ceny turnusu. Parametry vyjadřují v podstatě tu samou informaci ale v jiných výpočtech, které mají výsledky v různých řádech a jiný informační charakter.

Dosud byly identifikovány parametry ovlivňující optimalizační algoritmus a nyní budou analyzovány požadavky na parametry ovládání systému. Tyto parametry již nebudou za-

pisovány do konfiguračního souboru, ale budou přijímány pouze jako argumenty příkazové řádky. Nejdůležitějšími parametry budou dvě data určující časové rozmezí, odkdy dokdy se mají turnusy vytvořit. Data budou představovat první a poslední den tvorby turnusů. Dalším vhodným parametrem bude místo uložení konfiguračního souboru. Při práci se systémem bude nutné často experimentovat s nastavením parametrů algoritmu optimalizace. Místo uložení konfigurace bude vhodné například pro rychlé otestování nejúspěšnějších parametrů z předcházejících experimentů.

Analýza požadavků na parametry v tomto bodě končí. Není však vyloučeno, že v průběhu vývoje a užívání systému dojde k nalezení dalších vhodných parametrů. Při návrhu a pozdější implementaci systému musí být brán ohled na snadnou začlenitelnost dalších parametrů přebíraných od uživatelů. V tabulce (5.1) jsou přehledně uvedeny všechny požadované parametry.

5.2 Analýza udržovatelnosti systému

Ve druhé kapitole byly uvedeny požadavky kladené na tvorbu a vlastnosti turnusů. Tyto důvody je možné rozdělit do dvou kategorií. První kategorií jsou požadavky vytvářené legislativními nařízeními a druhou jsou požadavky ekonomické. Legislativní nařízení musí dodržovat všichni. Tyto požadavky se mění jednou za několik let a nedochází k zásadním změnám. Nebudou tedy předmětem zájmu každého běhu optimalizace. Starost o správu této části systému budou mít zkušení uživatelé schopní číst a upravovat tyto požadavky na úrovni zdrojového kódu. Při menších změnách v legislativě nemusí být potřeba upravovat zdrojové kódy. V kapitole 5.1.1 se v analýze parametrů systému vyskytují i parametry vztahující se k legislativním omezením. Bude-li změna legislativy spočívat v úpravě hodnot, bude snadné změnit odpovídající parametry v konfiguračním souboru.

Ekonomické požadavky na turnusy budou doménou upravovanou častěji. Každá firma má jistě své vlastní představy o tom, jak mají jejich turnusy vypadat. Některé firmy mohou například chtít turnusy maximální délky s tím, že se budou řidiči během turnusu střídat, jiné naopak můžou chtít vytvářet turnusy tak, aby je mohl vykonat jeden řidič. Další možným místem úprav může být výpočet vhodnosti turnusu. Stejně tak jako legislativní požadavky budou upravovat ekonomické požadavky odborní uživatelé, kteří jsou schopni číst a upravovat zdrojové kódy systému.

Pro popsané požadavky na turnusy je dostatečná úroveň udržovatelnosti na úrovni zdrojových kódu. Další možností by bylo vytvoření jednoduchého skriptovacího jazyka, kterým by bylo možné tyto výpočetní požadavky popsat. Toto řešení by bylo univerzální a nebylo by nutné pro každou změnu překládat podstatnou část systému. Nicméně tvorba interpretu rovnic přesahuje rámec této práce a výpočty bude nutné opravovat na úrovni zdrojových kódů.

Tabulka 5.1: Přehled všech parametrů.

Parametr	Popis
Pravděpodobnost křížení	pravděpodobnost s jakou bude docházet při reprodukci ke křížení
Pravděpodobnost mutace	pravděpodobnost s jakou bude docházet při reprodukci k mutacím
Velikost populace	počet jedinců v populaci
Velikost náhrady	počet jedinců ze staré populace nahrazovaných jedinci z nové populace
Počet generací	kolik generací má algoritmus evolvovat
Velikost tabu množiny	kapacita množiny zakázaných operací
Počet cyklů tabusearch	délka lokálního vyhledávání turnusů
Práh vhodnosti	hranice, nad kterou je turnus považován za vhodný
Délka dlouhé přestávky	doba, po kterou nesmí řidič řídit
Délka první přestávky	doba, po kterou nesmí řidič řídit
Délka druhé přestávky	doba, po kterou nesmí řidič řídit
Maximální délka turnusu	maximální doba mezi začátkem a koncem turnusu
Nejlepší délka turnusu	doba mezi začátkem a koncem turnusu
Penalizace prázdné jízdy	číslo udávající penalizaci za prázdnou jízdu
Penalizace čekání	číslo udávající penalizaci za čekání
Penalizace dlouhého turnusu	číslo udávající penalizaci za dobu nad nejlepší délkou turnusu
Penalizace krátkého turnusu	číslo udávající penalizaci za turnus kratší než nejlepší délka turnusu
Cena jednotky vzdálenosti	číslo udávající cenu za jednotku vzdálenosti ujetou dopravním prostředkem při vykonávání turnusu
Cena dopravního prostředku	cena za provoz jednoho dopravního prostředku
Cena jednotky čekání	cena za čas strávený čekáním při vykonávání turnusu
Počáteční datum	datum, od kterého se začnou vytvářet turnusy
Koncové datum	datum, do kterého se budou turnusy vytvářet

Kapitola 6

Návrh a popis optimalizačního systému

Vytvářený program bude kvůli potřebě úprav poměrně složitý. Proto již od začátku návrhu programu musí být kladen důraz na používání technik umožňujících snadnou modifikovatelnost a udržovatelnost. Hlavním cílem návrhu je vytvořit systém tak, aby jednotlivé části byly co nejméně závislé jedna na druhé. Snížením počtu závislostí bude mnohem jednodušší upravit konkrétní nevyhovující algoritmus, protože se sníží pravděpodobnost vzniku chyby v jiné části systému.

6.1 Průběh algoritmu optimalizace

Předtím než může být celý systém vhodně navrhnut, je nutné důkladně popsat činnost celého systému. Na základě popisu činnosti systému je dál jednoduché rozdělit systém do podsystému a dál se věnovat návrhu oddělených podsystémů. Na obrázku 6.1 je ve vývojovém diagramu schématicky znázorněna veškerá činnost systému při tvorbě turnusů.

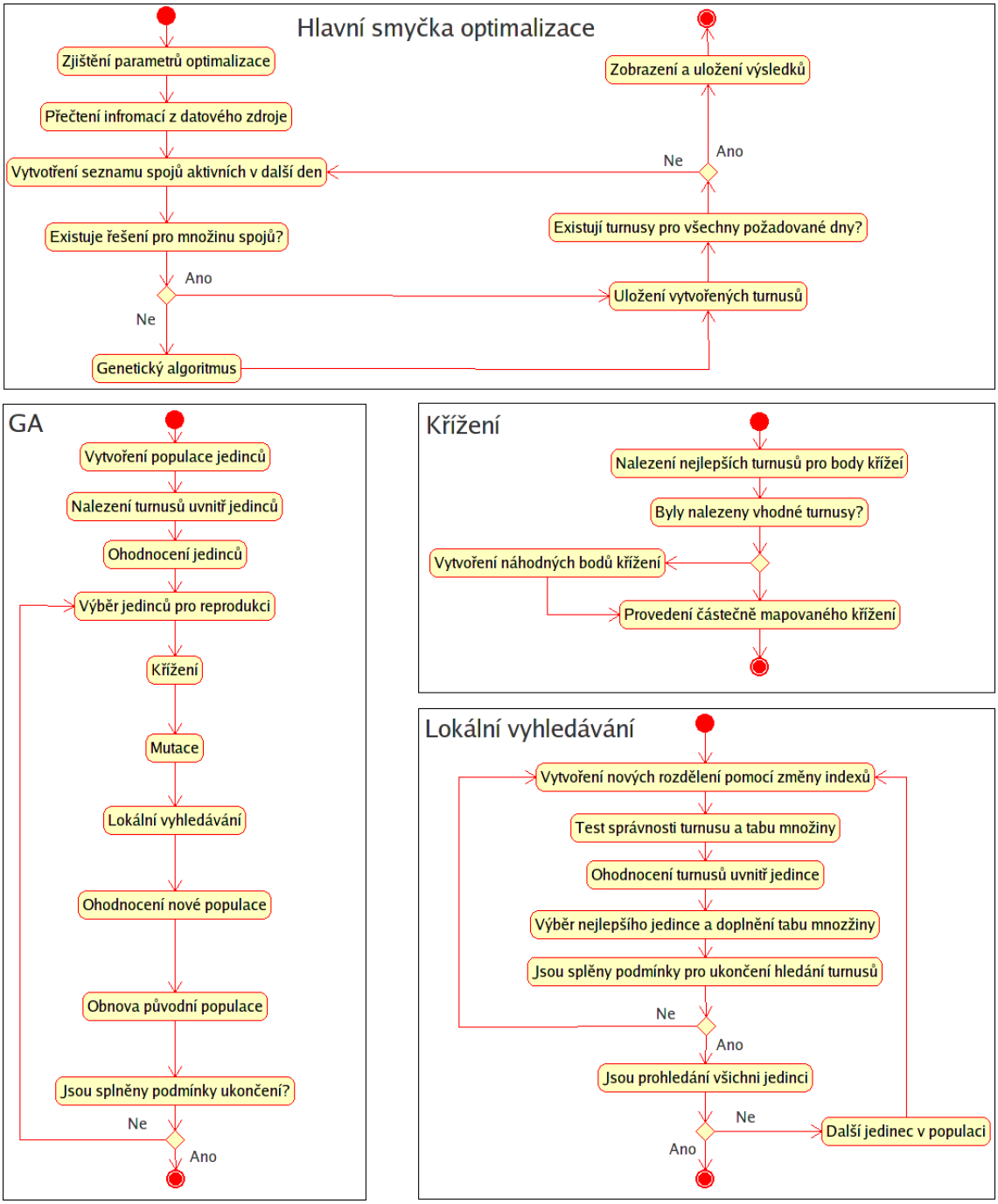
V předcházejících kapitolách byly popsán požadavky na systém. Analýza byla zaměřena převážně na parametry systému. Před zahájením tvorbu turnusů musí být všechny analyzované parametry zpracovány. Systém bude při zpracování parametrů postupovat následovně:

1. Zjistí, kde se nachází konfigurační soubor. Místo jeho uložení je buď v aktuálním adresáři nebo je zadáno argumentem na příkazové řádce.

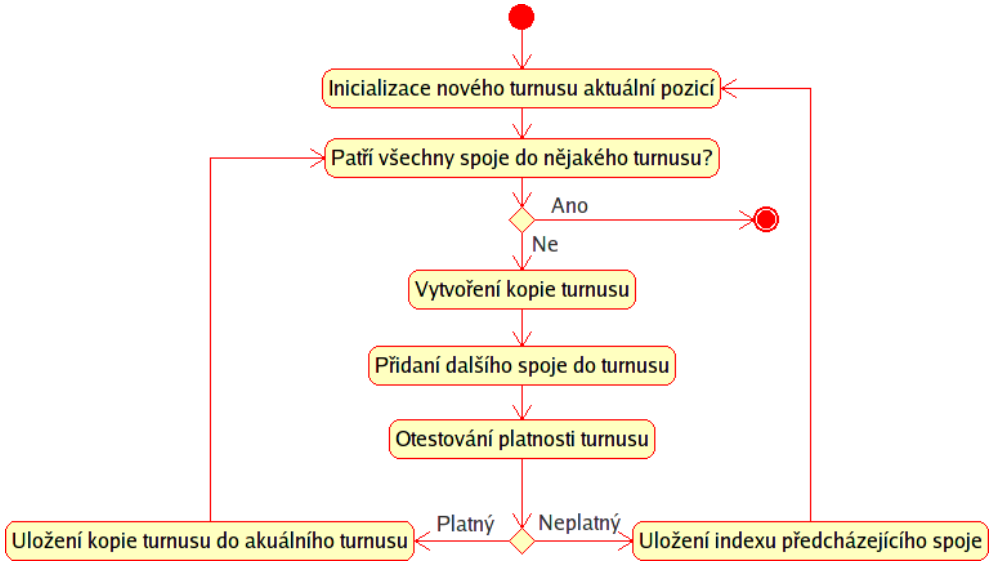
2. Po zjištění místa uložení konfiguračního souboru jsou načteny hodnoty parametrů a následně uloženy v paměti.

3. Systém pokračuje dalším zpracováním argumentů příkazové řádky. Hodnoty parametrů vyskytující se v argumentech jsou nahrazeny novými hodnotami.

4. Veškeré nastavení je zobrazeno uživateli pro potvrzení.



Obrázek 6.1: Vývojový diagram.



Obrázek 6.2: Vývojový diagram inicializace turnusů v chromozómu.

Po zpracování a potvrzení parametrů je vše připraveno pro zahájení tvorby turnusů. Z argumentů příkazové řádky získal systém informaci o požadovaném časovém rozmezí, pro které mají být turnusy vytvářeny. Tvorba turnusů bude probíhat v cyklu od začátku časového rozmezí do jeho konce. Pro každý den bude nalezena množina všech aktivních spojů (spojů obsluhovaných ten den) a tato množina bude předána algoritmu sestavujícího turnusy. Bylo by naivní nechat vytvářet turnusy znovu pro každou množinu spojů. Množina aktivních spojů je utvářena omezeními jako jsou například dny v týdnu, ve které je spoj aktivní. Po vytvoření turnusů pro první týden se již s vysokou pravděpodobností budou množiny aktivních spojů opakovat a nebude nutné znovu spouštět dlouhý a náročný algoritmus tvorby turnusů.

Algoritmus tvorby turnusů byl popsán v kapitole 4. Jeho průběh bude téměř stejný jako u všech genetických algoritmů. Na začátku dojde k vytvoření počáteční populace. Jedinci v této populaci budou vytvářeni pomocí náhodných permutací předané množiny aktivních spojů. Po vytvoření a inicializaci počáteční populace proběhne ohodnocení všech jedinců. Před ohodnocením jedinců bude provedeno nalezení rozdělení permutace spojů do turnusů. Z vytvořených turnusů bude dále vypočítána fitness hodnota jedince. Prvotní rozdělení do turnusů proběhne postupným přidáváním spojů do turnusu. Při každém přidání spoje proběhne test na porušení platnosti turnusu. Jakmile návratová hodnota testu označí turnus za neplatný, dojde k uložení předcházející pozice do seznamu konců turnusů a začne se tvořit nový turnus počínaje spojem, který zapříčinil porušení platnosti turnusu. Postup počáteční inicializace turnusů je znázorněn na obrázku 6.2. Podrobný popis hledání turnusů je uveden v kapitole 4.5

Po ohodnocení počáteční populace bude algoritmus probíhat podle obecného schématu

průběhu genetických algoritmů. Nejdříve dojde k výběru jedinců, jako rodičů budoucí populace. Pro selekční schéma bude použito kolo štěstí. Po vytvoření množiny rodičů budou vybrány dvojice jedinců, které se s určitou pravděpodobností zkříží. S pravděpodobností 1-pravděpodobnost křížení dojde pouze k vytvoření kopie předků. Po křížení vznikne množina nových jedinců, u kterých bude dále s jistou pravděpodobností provedena mutace. Po dokončení reprodukce musí být noví jedinci nejdříve ohodnoceni a následně bude stávající populace obnovena nově vytvořenými jedinci. Ohodnocování nových jedinců bude probíhat naprosto stejně jako ohodnocovaní počáteční populace. Takto bude algoritmus vytvářet nové generace populací tak dlouho, dokud nedosáhne předem stanoveného počtu nových generací. V každé generaci bude nalezen nejlepší jedinec a jako výsledek bude vrácen jedinec nejlepší za celý průběh algoritmu.

Po dokončení tvorby turnusu pro množinu aktivních spojů bude výsledek uložen společně se dnem a původní množinou spojů. V další iteraci cyklu přes dny bude možné nalézt pomocí porovnání původní množiny spojů s množinou aktuální stejný případ a výsledku se přiřadí aktuální datum, pro který jsou turnusy vytvořeny. Po vytvoření spojů pro poslední den bude snadné zobrazit uživateli přehledné statistiky o tom, jaké turnusy byly vytvořeny a které dny mají shodné turnusy.

6.2 Vstupy a výstupy

Jedinými vstupy systému budou argumenty příkazové řádky a soubor s parametry optimalizace. Formát argumentů příkazové řádky bude standardního tvaru. Argumenty příkazové řádky ve standardním tvaru mají před jménem jednu nebo dvě pomlčky a za jménem hodnotu nebo prázdné místo. Obvykle se pro argumenty používají dvě jména. První jméno je dlouhé, vystihuje význam argumentu a píší se před něj dvě pomlčky. Druhé jméno bývá mnohem kratší a tvoří ho například jedno charakteristické písmeno z prvního jména. Před krátká jména argumentů se píše pouze jeden znak -. V argumentech příkazové řádky se dále mohou vyskytovat pouze hodnoty nepatřící k žádnému jménu. Tyto hodnoty se musí vyskytovat pouze v přesně daném pořadí tak, aby systém rozpoznal význam každé této nepojmenované hodnoty. Pro argumenty se stejným významem jako parametry v konfiguračním souboru budou použita dlouhá jména shodná se jmény parametrů v konfiguračním souboru.

Vstupní soubor s parametry optimalizace může mít mnoho různých formátů. Pro strojové zpracování je například vhodné používat XML. Ve vývojových knihovnách je poměrně dobře podporovaným formát INI souborů. INI soubory jsou prosté textové soubory obsahující na každém řádku jméno jednoho parametru a jeho hodnotu. Na řádku lze tedy nalézt textovou hodnotu odpovídající jménu parametru. Za jménem se musí nacházet znak = a poté následuje hodnota parametru. Kromě parametrů s hodnotami se na řádcích může vyskytovat několik dalších základních struktur, ale ty nebudou v tomto projektu použity.

Systém bude mít dva druhy výstupů. Prvním druhem jsou informační výstupy. Tyto

výstupy mají za úkol spravovat uživatele o průběhu optimalizace. Jako výstup mají hodnotu pouze dočasnou. Po dokončení optimalizace je možné tyto výstupy zlikvidovat, případně je možné tyto výstupy přímo zakázat. Podsystém pro správu zdrojů bude poskytovat zdroj, do kterého budou informace zapisovány. Jako nejvhodnější umístění těchto výstupů se jeví standardní výstup. Druhým typem výstupů jsou již úplné výsledky optimalizace. Jejich hodnota je po ukončení optimalizace na rozdíl od informačních výstupů obrovská. Tyto výsledky je nutné po dokončení optimalizace co nejlépe archivovat. Stejně tak jak pro informační výstupy bude podsystém zdrojů poskytovat úložiště i pro tento typ výstupů.

Výsledky optimalizace musí být člověkem snadno interpretovatelné a počítačem zpracovatelné. Ve výsledcích musí být pro každý den jasně uvedena sekvence identifikátorů spojů definující turnus. Těchto sekvencí identifikátorů bude tolik, kolik bude pro daný den nalezeno turnusů. Jelikož se množiny spojů aktivních v jeden den budou často opakovat, není potřeba uvádět množinu turnusů ke každému dni zvlášť. Naopak bude k jedné množině turnusů uveden seznam dní, ve kterých bude použita. Takovýto výstupní formát bude strojově snadno zpracovatelný, ale neobsahuje žádné důležité informace. Uživatelé budou chtít vidět vlastnosti výsledné množiny turnusů. Ve výstupu se tak budou nacházet ještě informace o počtu turnusů, výsledné ceně, době čekání, prázdných přejezdech a další. Nyní je těžké určit úplnou množinu informací, které budou uživatele zajímat. Systém musí být připraven na možnost rozšíření množiny poskytovaných informací.

V tabulce 6.1 jsou uvedeny příklady všech vstupních a výstupních formátů.

Tabulka 6.1: Ukázka zpracovávaných formátů.

Typ	Formát
Argumenty příkazové řádky	–populationSize 100 -o results.txt
Konfigurační soubor	mutationProb = .8
Informační výstupy	libovolné textové hlášení na standardním výstupu
Výsledky	2009-05-26 count=2, price=26000 $< A1, A2, A3, A4 >$ $< A5, A6, A7, A8 >$

6.3 Rozdělení do podsystémů

V předcházejícím popisu činnosti systému je možné vysledovat čtyři hlavní části činnosti systému. Tři z nich jsou skutečně rozpoznatelné a poslední čtvrtá je nezbytná po každý systém. Vytvořené rozdělení do podsystémů respektuje moderní zásady návrhu systému s cílem snížit počet závislostí na minimum [7]. Znázornění podsystémů a interakcí mezi nimi je vidět na obrázku 6.3

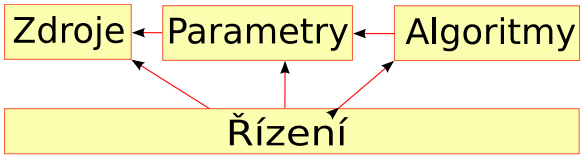
Správa parametrů Centralizovaná správa a zpracování parametrů z konfiguračního sou-

boru a příkazové řádky. Parametry ze jmenovaných zdrojů získá, sjednotí hodnoty parametrů vyskytujících se v obou zdrojích a dále tyto parametry uchovává v paměti po celou dobu běhu optimalizace a zpřístupňuje ostatním částem systému. Je schopná poskytnout informace o podporovaných parametrech a stejně tak je možné získat přehled všech parametrů s jejich hodnotami.

Správa zdrojů Připojuje se k databázi nebo jinému úložišti odkud je možné získávat množinu aktivních spojů nebo zjistit vzdálenost mezi jednotlivými stanicemi. Správa zdrojů také poskytuje zdroje, do kterých je možné uložit výsledky optimalizace nebo posílat informace o aktuálním průběhu výpočtu. Pomocí vysoké úrovně abstrakce je možné pružně měnit datová úložiště, aniž by bylo třeba zasahovat do ostatních systémů.

Algoritmy Pomocí rozhraní a vysoké úrovně abstrakce poskytuje systému generický přístup k algoritmům. Svým přístupem umožňuje jednoduchou změnu algoritmů projevující se v celém systému. Tento podsystém obsahuje například algoritmus pro výpočet vhodnosti turnusu nebo fitness funkci pro genetický algoritmus.

Řízení Poslední podsystém využívající všechny ostatní podsystémy pro splnění požadavků zadaných uživatelem. Zajistí inicializaci všech ostatních podsystémů a uvede do chodu optimalizační proces. Po dokončení optimalizací poskytne uživateli informace o průběhu a uloží výsledky.



Obrázek 6.3: Znázornění subsystémů a jejich vzájemných vazeb.

6.4 Objektový model

V souladu s požadavky a analýzou je dále navrhnut objektový model systému. Při návrhu objektového modelu jsou dodržovány moderní postupy pro vytvoření elegantního a snadno udržovatelného systému[7]. Návrh objektového modelu se snaží vytvářet systém na nejvyšší možné úrovni abstrakce. Pro dosažení požadované úrovně abstrakce jsou hojně používány rozhraní a návrhové vzory. S využitím návrhových vzorů je možné snadno dosáhnout udržovatelné a přehledného modelu. Objektový model také počítá s využitím vývojářských knihoven.

6.4.1 Základní třídy

Celý systém je rozdělen do čtyř logických celků. Tyto celky budou mezi sebou různě komunikovat. Jejich komunikace bude prováděna prostřednictvím objektů a volání metod. Základní třídy budou tvořit právě tu množinu objektů využívaných pro společnou komunikaci. Základní třídy jsou knihovnou tříd, které jsou v systému velmi hojně využívány. Tato knihovna však nebude obsahovat pouze třídy, ale také mnoho rozhraní. Rozhraní z knihovny základních tříd jsou využívány ze všeho nejvíce. Pomocí nich bude dosaženo maximálního stupně abstrakce a tím snadné modifikovatelnosti celého systému.

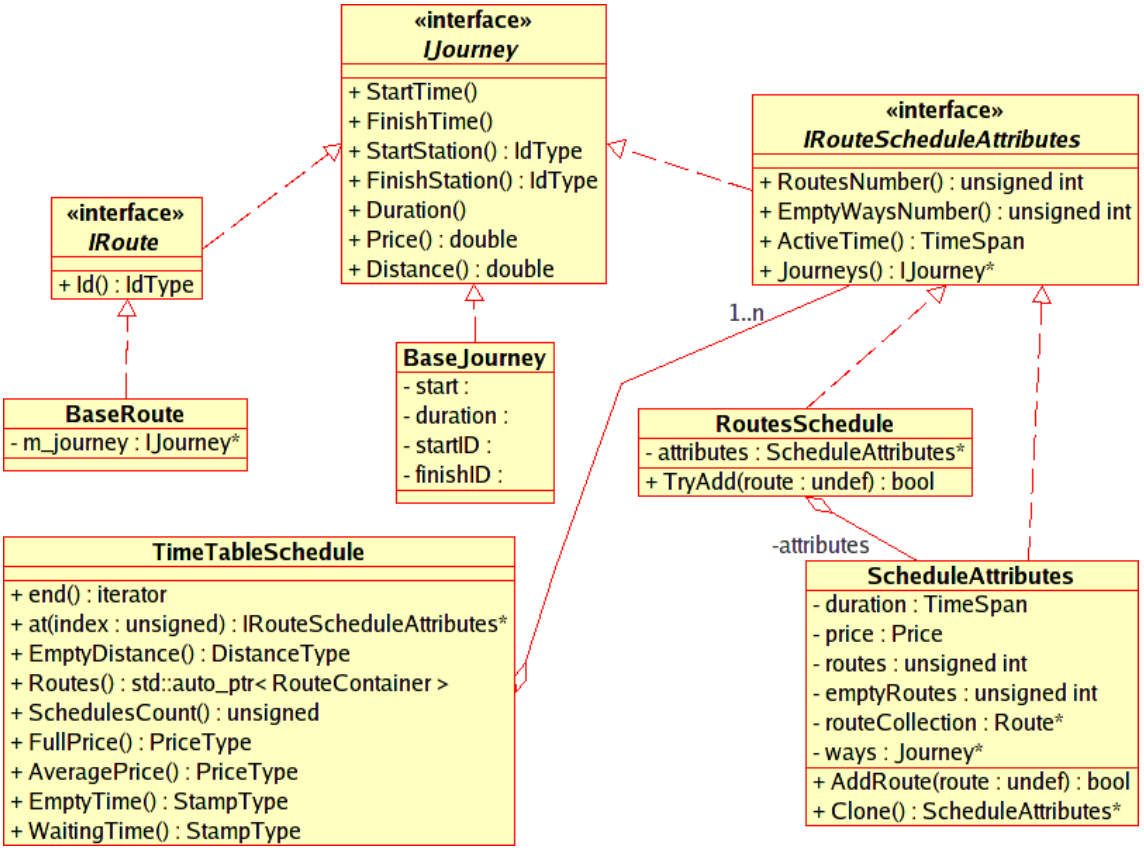
Knihovna systému obsahuje definice tříd a rozhraní vztahující se převážně k jízdním řádům. Na obrázku 6.4 je vidět diagram tříd popisující cesty, spoje i turnusy.

`IJourney` je rozhraní popisující cestu mezi dvěma stanicemi. Rozhraní prostřednictvím veřejných metod poskytuje veškeré důležité informace o cestě.

`IRoute` dědí z rozhraní `IJourney` a představuje jeden spoj. Každý spoj je v podstatě jedna cesta, která se dle stanovených pravidel opakuje. Během obsluhy spoje může dopravní prostředek několikrát zastavit, ale při tvorbě turnusů je nutné znát pouze výchozí a cílovou stanici. Každý spoj musí mít unikátní identifikátor a jeho hodnotu je možné získat jako návratovou hodnotu volání metody *Id()*.

`IRouteScheduleAttributes` je další entitou dědící z rozhraní `IJourney` a představuje atributy turnusu, protože turnus je pouze jedna cesta dopravního prostředku mezi výchozí stanicí prvního spoje a cílovou stanicí posledního spoje. Kromě vlastností cesty poskytuje další užitečné informace o turnusu, jako je počet spojů, počet prázdných cest, délka užitečně stráveného času a nakonec seznam všech cest. Seznam všech cest turnusu obsahuje jak spoje, tak cesty mezi cílovými a výchozími stanicemi spojů, které na sebe v turnusu navazují, ale tyto stanice mají rozdílné.

`RouteSchedule` je třída implementující rozhraní `IRouteScheduleAttributes` a představuje jeden turnus. Instance třídy `RouteSchedule` bude obsahovat objekt třídy `ScheduleAttributes` a většinu volání metod, které dědí z rozhraní atributů, pouze přepošle vnitřnímu objektu. Tento model spolupráce objektů je zvolen pro snadnou implementaci přidávání nových spojů do turnusu. Přidání spoje do turnusu se bude vykonávat metoda *TryAddRoute()*, která vrátí logickou hodnotu pravdy, pokud se podaří spoj přidat do turnusu. Pokud nebude možné spoj do turnusu přidat, vrátí metoda hodnotu logické nepravdy. Spoj není možné do turnusu přidat, pokud poslední spoj turnusu nemůže stihnout začátek přidávaného spoje nebo by nově vzniklý turnus porušoval pravidla validity turnusu. Přidání proběhne tak, že se vytvoří kopie vnitřního objektu, do kterého se přidá další spoj. Tento objekt se předá metodě kontrolující validitu turnusu a ta rozhodne o jeho validitě. Pokud bude turnus v kopii validní, původní vnitřní objekt se zahodí a novým vnitřním objekt se stane kopie, jinak se zahodí kopie.



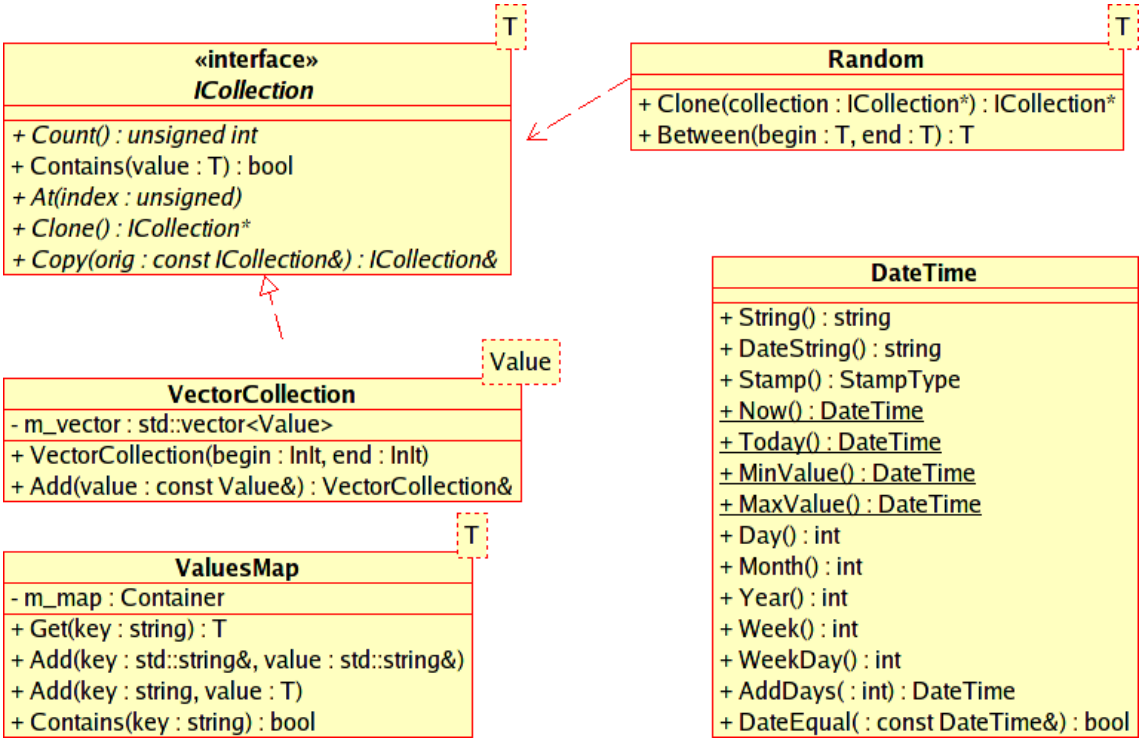
Obrázek 6.4: Diagram tříd jízdních řádů.

`TimeTablesSchedule` je kontejner pro turnusy. Tato třída bude představovat systém turnusů pro jeden den. Poskytuje metody se zajímavými informacemi o turnusech a také umožňuje sekvenční přistup k všem uloženým turnusům. Rozhraní třídy je navrženo tak, aby bylo částečně kompatibilní s STL kontejnery. Pro práci s objekty této třídy tak bude možné využívat STL algoritmy a další šablony pracující s těmito kontejnery.

Nakonec diagram tříd pro spoje obsahuje základní implementaci rozhraní cesty a rozhraní spoje. Implementace rozhraní cesty není úplné. Obsahuje pouze ty atributy, u kterých se předpokládá, že budou ve všech implementacích stejné. Tato implementace slouží pouze jako bázová třída určená pro vytváření potomků. Bázová třída urychlí implementaci konkrétních tříd popisujících cestu tím, že se vývojář může zaměřit pouze na ty metody, u kterých se předpokládají rozdílné požadavky na jejich implementaci. Implementace rozhraní spoje je úplná a využívá vnitřního objektu implementujícího rozhraní `IJourney`, na který směruje všechna volání metod zděděných z rozhraní `IJourney`.

Po třídách pro spoje dále knihovna obsahuje různé pomocné třídy, které jsou vidět v diagramu na obrázku 6.5.

`ICollection` rozhraní pro konstantní uspořádané kolekce hodnot. Poskytuje metody pro



Obrázek 6.5: Diagram pomocných tříd.

zjištění, zda obsahuje požadovanou hodnotu a pro získání prvku na určitém indexu.

`VectorCollection` je implementace rozhraní kolekce používající jako vnitřní reprezentaci STL vector.

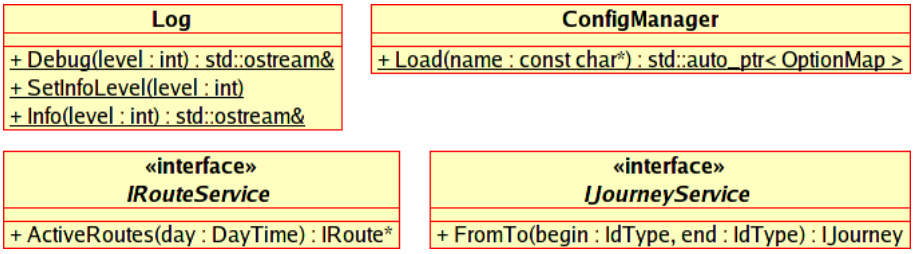
`Random` jmenný prostor obsahující metody s náhodným chováním. *Between()* vrací náhodnou hodnotu v požadovaném rozsahu. *Clone()* vytváří kopii kolekce, ve které budou původní prvky na jiných pozicích.

`DateTime` je třída pro práci s datem a časem. Poskytuje statické metody pro získání aktuálního času a dnešního data. Tyto hodnoty mají stejné datum, ale aktuální datum má na rozdíl od aktuálního času nastaveny hodiny, minuty a sekundy na hodnotu 0. Dále poskytuje metody pro konverzi data do řetězcové podoby a zpět.

`ValuesMap` je asociativní pole pro předávání proměnlivého počtu argumentů. Hodnota argumentu je asociována s řetězcem tvořícím klíč. Na základě klíče je možné hodnoty v poli měnit a také je z něj získávat.

6.4.2 Správa zdrojů

Správa zdrojů je popsána dvěma třídami a dvěma rozhraními. Obě třídy ve správě zdrojů jsou statické. Jde spíše o knihovní třídy. Tyto třídy nebude nikdy potřeba instanciovat.



Obrázek 6.6: Diagram tříd správy zdrojů.

Obsahují pouze statické metody. V krajním případě by bylo možné tyto třídy nahradit jmennými prostory, ale tyto prostory by obsahovaly malý počet funkcí, a proto je vhodnější umístit funkčnost do statických metod. Na obrázku 6.6 je zobrazen diagram tříd pro podsystém správy zdrojů.

`Log` je statická třída poskytující logovací STL proudy (streamy). Poskytuje proudy nejen pro komunikaci s uživateli, ale také proudy pro ladící výpisy. Popsané metody přebírají jako jediný argument celé číslo udávající priority zprávy, která má být do proudu zapsána. Metody podle tohoto argumentu a nastavené úrovně vrací skutečný proud. Pokud priorita zprávy nedosahuje nastavené úrovně, potom vrací prázdný proud. Priorita zpráv je inverzní, tedy čím větší číslo, tím menší priorita.

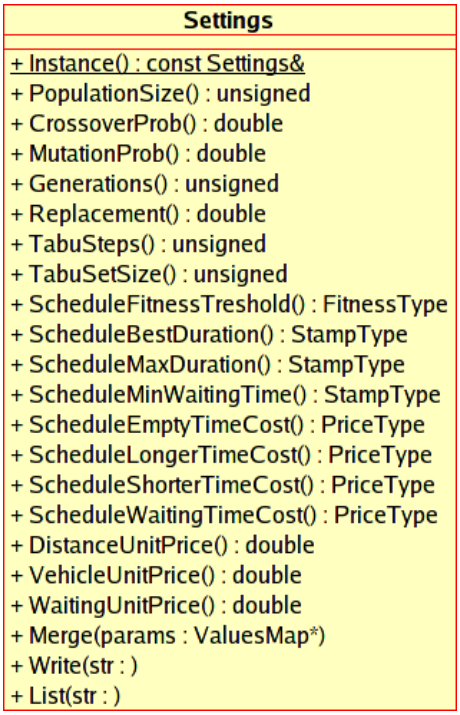
`ConfigManager` je statická třída s metodou pro načítání a zpracování konfiguračních souborů. Metoda *Load()* přebírá jméno požadované konfigurace a vrací `ValuesMap` s hodnotami z konfiguračního zdroje. Jméno konfigurace se předává bez přípon a případných cest v souborovém systému. Implementace metody sama určuje, kde se bude konfigurační zdroj vyhledávat.

`IRouteService` je rozhraní pro zdroj, který umožňuje získání spojů aktivních v požadovaný den. Metodě *ActiveRoutes()* je předána instance třídy `DateTime` a v návratové hodnotě bude množina aktivních spojů.

`IJourneyService` je rozhraní pro zdroj, který poskytuje vyhledávání cest mezi dvěma stanicemi. Metodě *FromTo()* jsou předány dva identifikátory stanic a v návratové hodnotě bude instance rozhraní `IJourney` popisující cestu mezi zadanými stanicemi.

6.4.3 Správa parametrů

Spravovat parametry bude jediná třída. Její implementace bude provedena dle návrhového vzoru singleton[8], čímž bude přístup k parametrům centralizován do jediného místa. Na obrázku 6.7 je diagram s třídou nastavení `Settings`. Dle návrhového vzoru poskytuje třída statickou metodou vracející jedinou instanci. Před prvním zavoláním této metody dojde k inicializaci instance, která je následně vrácena jako návratová hodnota volání. Objekt



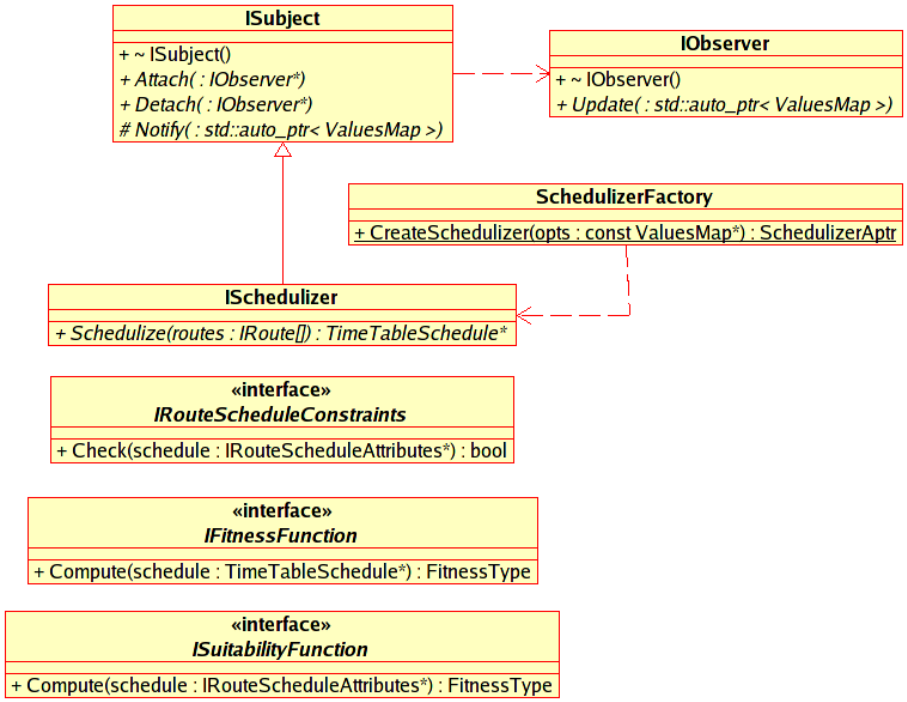
Obrázek 6.7: Diagram třída nastavení.

třídy `Settings` kromě metod vracejících hodnoty parametrů poskytuje metody pro správu obsahu. Pomocí metody *Merge()* jsou z argumentu *ValuesMap* přenastaveny podporované parametry. Metody *Write()* a *List()* slouží pro výpis obsahu instance a pro výpis seznamu všech zpracovávaných parametrů.

6.4.4 Algoritmy

Diagram tříd podsystému algoritmů (obrázek 6.8) je tvořen převážně rozhraními. Jediná entita, která není v diagramu rozhraní, je statická třída `SchedulizerFactory` s metodou implementovanou dle návrhového vzoru tovární metoda[8]. Tovární metoda vrací instance rozhraní `ISchedulizer`, což jsou implementace algoritmu provádějícího tvorbu systému turnusů z množiny aktivních spojů. Pro možnost sledování průběhu optimalizace jsou vytvořeny dvě rozhraní dle návrhového vzoru pozorovatel. Jedná se o rozhraní `ISubject` pro objekty poskytující informace a rozhraní `IObserver` pro objekty, které poskytnuté informace zpracovávají. Z rozhraní `ISubject` dědí rozhraní `ISchedulizer` a při vhodné implementaci rozhraní `IObserver` bude snadné zobrazovat uživatelům podrobné informace o průběhu výpočtu.

Další rozhraní již popisují důležité výpočty systému. Pro genetický algoritmus je vytvořeno rozhraní `IFitnessFunction` a rozhraní `ISuitabilityFunction`. První rozhraní vytváří abstrakci pro funkci počítající fitness hodnotu systému turnusů a druhé pro výpočet vhodnosti jednoho turnusu. Poslední rozhraní `IRouteScheduleConstraints` popisuje



Obrázek 6.8: Diagram tříd podsystému algoritmů.

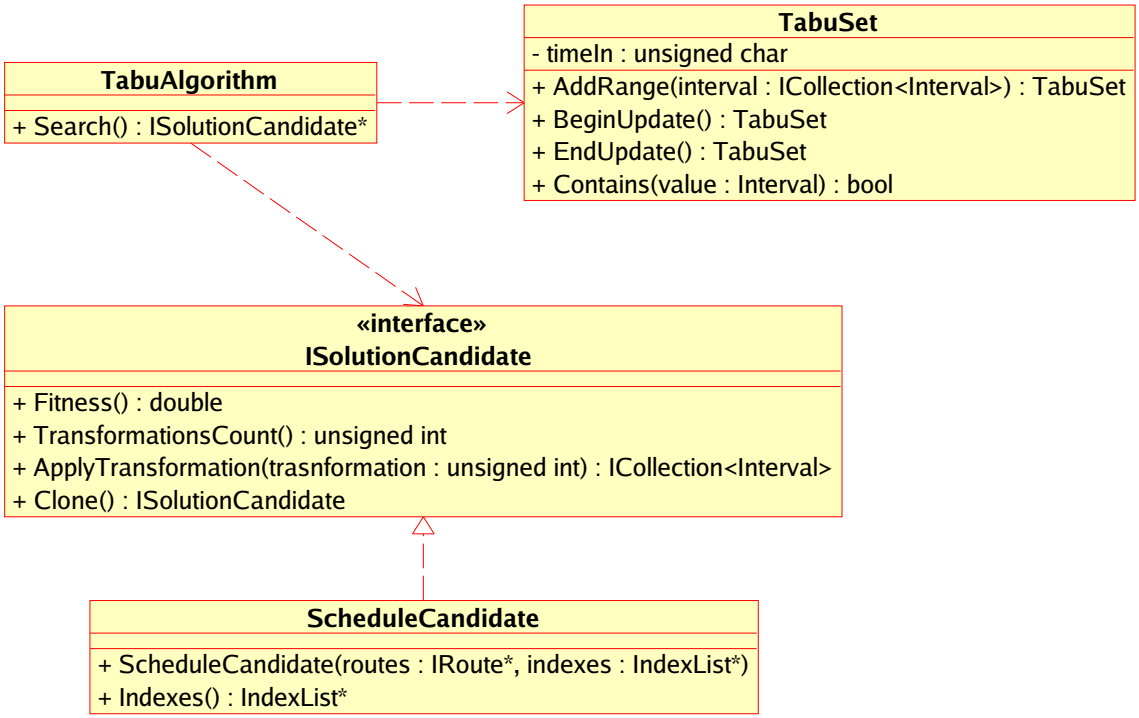
kontrolu podmínek omezujících platnost jednoho turnusu.

6.4.5 Tabu search

Objektový model pro metodu lokálního prohledávání chromozómu tabu search respektuje popis algoritmu v kapitole 3.2. Metodu zakázaného hledání reprezentuje třída `TabuSearch` s metodou *Search()*, která přebírá jako argument objekt rozhraní `ISolutionCandidate`. Na obrázku 6.9 je zobrazen diagram tříd zakázaného hledání.

`TabuSet` je třída reprezentující množinu zakázaných operací. Omezení velikosti množiny nebude provedeno omezením kapacity, ale omezením času stráveného uvnitř množiny. Dle popisu v předcházejících kapitolách bude množina obsahovat seznam intervalů změněných aplikováním transformace. Jedno aplikování transformace může vyvolat $0..N$ změn, kde N je počet všech intervalů uvnitř chromozómu. Provedením jedné transformace by mohla být zcela zaplněna kapacita množiny zakázaných operací. Řešením tohoto problému je nastavení času, po jehož uplynutí budou intervaly vymazány. Čas bude logický a bude představovat počet provedení dvojice metod *BeginUpdate()* a *EndUpdate()*. Každé provedení této dvojice metod bude znamenat posun času.

`ISolutionCnadidate` je rozhraní představující řešení problému. Rozhraní je navrženo tak, aby bylo možné při případné změně reprezentace problému zachovat implementaci zakázaného prohledávání. Rozhraní obsahuje metody pokrývající potřeby algoritmu



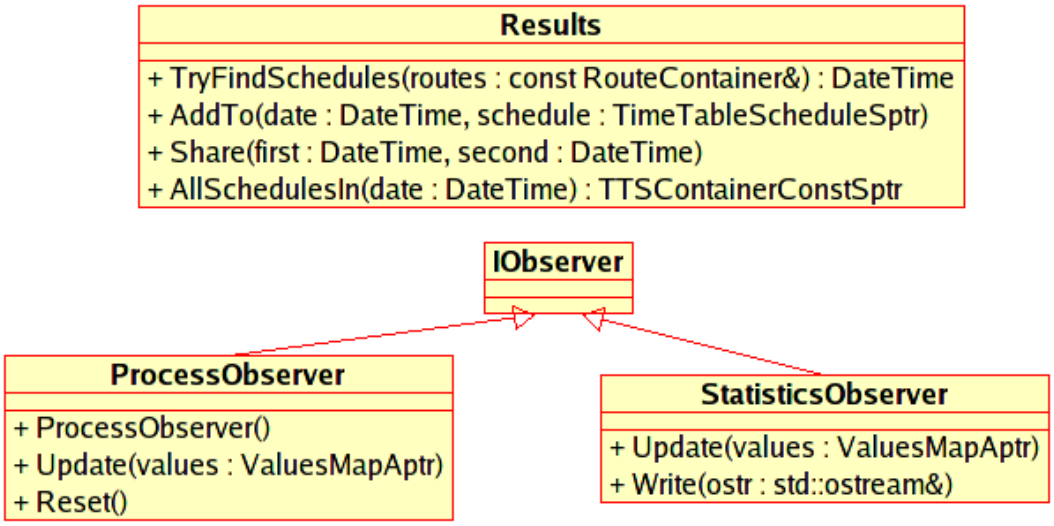
Obrázek 6.9: Diagram tříd zakázaného hledání.

zakázaného hledání v nejvyšší možné úrovni abstrakce. Tím je získán potřebný prostor pro manipulaci s reprezentací řešeného problému. *TransformationsCount()* je metoda poskytující počet možných operací a metoda *ApplyTransformatino()* provádějící jednu z transformací. Metoda *ApplytTrasnformation()* bude přijímat jako jediný argument celé číslo v rozsahu $< 0, TransformationsCount() - 1 >$, který bude představovat indentifikátor prováděné operace.

`ScheduleCandidate` je implementace rozhraní `ISolutionCandidate` a s metodami implementovanými tak, aby jejich chování odpovídalo reprezentaci problému popsané v předcházejících kapitolách. Metoda *TrasnformationsCount()* bude vracet dvojnásobek počtu indexů uvnitř chromozomu. Dvojnásobek počtu indexů je vrácen proto, že každý index je možné buď o jedničku zvětšit nebo zmenšit. Index, který se má změnit, získáme jako výsledek celočíselného dělení argumentu dvojkou $index = arg/2$. Dále pokud bude argument sudý, bude hodnota indexu dekrementována, jinak bude inkrementována.

6.4.6 Řízení

Podsystém řízení je poměrně jednoduchý. Veškerá důležitá činnost je implementována v ostatních podsystémech a řízením pouze provede inicializaci a spustí optimalizační algoritmus. V řídícím podsystému je deklarováno několik tříd, které vesměs implementují rozhraní



Obrázek 6.10: Diagram tříd řídícího podsystému.

z ostatních podsystémů. Nejvíce práce vykonávají implementace rozhraní `IObserver` popsané v kapitole 6.4.4 a zobrazené na obrázku 6.8. Diagram tříd řízení je zobrazen na obrázku 6.10.

`Results` je kontejner na výsledky. Do instance této třídy budou přidávány výsledky optimalizačního algoritmu. Každý výsledek bude spojen s konkrétním datem, pro který byl vytvořen, a množinou spojů, ze které byl vytvořen. Množina spojů odpovídá množině spojů aktivní v přiřazeném datu. Před zahájením optimalizačního algoritmu bude metodou *TryFindSchedules()* provedeno vyhledání množiny aktivních spojů v množinách, pro které již existují výsledky. Pokud bude výsledek nalezeni, provede se pomocí metody *Share()* sdílení výsledků mezi daty.

`ProcessObserver` implementuje rozhraní `IObserver` a slouží pro informování uživatelů o průběhu výpočtu.

`StatisticsObserver` sbírá statistiky a je schopný tyto statistiky zapsat do STL streamu.

Kapitola 7

Popis implementace

Systém popsaný v předcházejících kapitolách byl implementován v jazyce C++ s využitím dvou velmi užitečných knihoven **GAlib** a **Boost**. Při implementaci systému byl neustále kladen důraz na snadnou udržovatelnost a přenositelnost[1].

7.1 GAlib

GAlib je knihovna implementovaná v programovacím jazyce C++ a obsahuje mnoho komponentů genetických algoritmů[11]. Je snadno použitelná a jednoduše rozšiřitelná. Díky tomu je možné genetické algoritmy používat ve všech programech. Její použití je motivováno také tím, že je díky svému rozšíření hojně otestovaná a proto bude obsahovat minimum logických chyb, které by se mohly vyskytnout ve vlastní implementaci. Výhodou této knihovny je také to, že funguje jak na Unixových systémech, tak i na platformě Windows.

7.2 Boost

Knihovna Boost je sjednocením mnoha volně dostupných C++ knihoven, které jsou vytvořeny experty z celého světa[10]. Její vlastnosti nejlépe popisuje citát z knihy C++ coding standards od autorů Herb Sutter a Andrei Alexandrescu: "...one of the most highly regarded and expertly designed C++ library projects in the world."[5].

Potenciál knihovny Boost je nesmírný, ale v tomto projektu je z jejího potenciálu využito naprosté minimum. Ovšem i toto minimum zjednodušilo vývoj. Největším přínosem knihovny jsou inteligentní ukazatele. C++ je jazyk, ve kterém si programátoři musí sami spravovat paměť. To dělá jazyk C++ velmi efektivním, ale na druhou stranu může znepříjemňovat život. Chytré ukazatele vyřeší většinu problému se správou paměti a programátor se může věnovat podstatným částem implementace.

7.3 Důležité algoritmy

V této sekci jsou popsány některé důležité algoritmy, jejichž implementace je nutné důkladněji vysvětlit nebo doposud nebyly v žádné z kapitol popsány. Některé důležité algoritmy jsou popsány v kapitole 6.4 popisující objektový model. S algoritmy, jejichž princip je důkladně rozebrán v předcházejících kapitolách a byly implementovány shodně s jejich popisem, se nebudeme dále zabývat.

7.3.1 Genetický algoritmus

Genetický algoritmus je implementován s pomocí knihovny GAlib. Je použit steady state genetický algoritmus což znamená, že v průběhu evoluce existuje populace o jisté velikosti a v každé generaci je novými jedinci nahrazen pouze předem stanovený počet jedinců z generace původní. Velikost populace bude možné nastavit pomocí parametru. Hodnota parametru bude udávat konstantu, s jejíž pomocí se z velikosti množinu turnusů vypočte velikost populace. Počet nahrazovaných jedinců je také možné měnit parametrem, který bude udávat procentuální podíl z velikosti populace. Pro selekční schéma je použito výchozí schéma tohoto modelu, což je kolo štěstí. Genetickému algoritmu je dále možné pomocí parametrů nastavit pravděpodobnost mutace a křížení. Délka evoluce je omezena pomocí počtu evolvovaných generací a i tato hodnota je nastavitelná parametrem. Fitness jedinců v generaci je sumou cen turnusů. Výpočet ceny turnusu je uveden v článku [4].

7.3.2 Operátor křížení

Podrobný postup při křížení je popsán v kapitole 4.3. Při implementaci bylo křížení rozšířeno o možnost získání více bodů křížení. Ve standardním pojetí operátoru PMX se náhodně vyberou dva body křížení. V popisu operátoru křížení je uvedeno, že body křížení se nebudou nalézat náhodně, ale vyberou se ty body, které tvoří nejvhodnější turnus. Je zřejmé že chromozóm může obsahovat více stejně ohodnocených turnusů, jejichž hodnota vhodnosti bude nejvyšší. Poté by nastal problém s výběrem jednoho turnusu z množiny nejlépe ohodnocených turnusů. Tento problém by se dal vyřešit například náhodným výběrem jednoho turnusu. V systému je operátor křížení implementován tak, že se použijí indexy všech vhodných turnusů.

V originální implementaci se po výběru bodů křížení přenesou geny mezi body křížení do nově vytvářeného jedince na stejné pozice jako v předkovi. Při provádění upraveného křížení se do nově vytvářeného jedince přenesou geny ležící uvnitř vhodných turnusů opět na svá místa. Takto implementovaný operátor křížení převede jeden souvislý úsek genů na několik disjunktních úseků.

Počet nalezených turnusů, které budou označeny jako vhodné, bude možné měnit pomocí parametru pojmenovaného jako "scheduleTreshold"(práh vhodnosti turnusu). Vhodnost turnusu je v aktuální podobě vypočítána rovnicí $max[(t - t_{nejlepší}) * c_{delší}, (t_{nejlepší} - t) * c_{kratší}] + t_{čekání} * c_{čekání} + t_{prázdné} * c_{prázdné}$, kde t je délka turnusu, $t_{nejlepší}$ je parametr

určující nejlepší délku turnusu, $c_{delší}$ je parametr určující cenu času přesahující nejlepší délku turnusu, $c_{kratší}$ je parametr určující cenu nevyužitého času , $t_{čekání}$ je čas čekaní při vykonávání turnusu, $c_{čekání}$ je parametr určující cenu za čekání, $t_{prázdné}$ je čas jízdy na prázdno a $c_{prázdné}$ je cena času na jízdy na prázdno. Uživatelé nastavují veškeré parametry a mohou si vypočítat hodnotu pro ně přijatelného turnusu. Rovnice se snaží respektovat legislativní omezení na dobu řízení řidičů. Při použití této rovnice se nepočítá s možností výměny řidičů během turnusu. Systém se snaží vytvářet turnusy na míru maximální doby řízení řidičů. Parametr určující nejlepší délku turnusu by měl být nastaven na 9 hodin. Řidiči mohou řídit i 10 hodin dvakrát týdně, ale je výhodnější, aby hodnotu 9 hodin nepřekračovali. Překračování 9 hodin může vést k neúnosnému zvyšování potřebného počtu řidičů.

7.4 Použití programu

Pro použití je nutné program nejprve sestavit. Pro správné sestavení je nutné, aby v hostitelském operačním systému byl nainstalován překladač **gcc** ve verzi vyšší než 4.0, automatický sestavovací systém **GNU/make** a přenositelný sestavovací systém **CMake**. Sestavení systému se provádí na Unixových operačních systémech příkazem *make* v kořenovém adresáři projektu. Po bezchybném sestavení je systém připraven ke spuštění. Před jeho spuštěním je ještě vhodné upravit konfigurační soubor umístěný v adresáři config/ a pojmenovaný settings.ini. Konfigurační soubor je optimalizován pro testovací množinu spojů a není nezbytně nutné ho upravovat. Testovací množina spojů je v systému zabudována, pokud dojde k sestavení beze změny parametrů sestavení.
Získání nápovědy systému je možné pomocí příkazu :

```
bin/routop -h
```

Pro spuštění optimalizace testovací množiny turnusů stačí zadat příkaz :

```
bin/routop -c bin
```

Kapitola 8

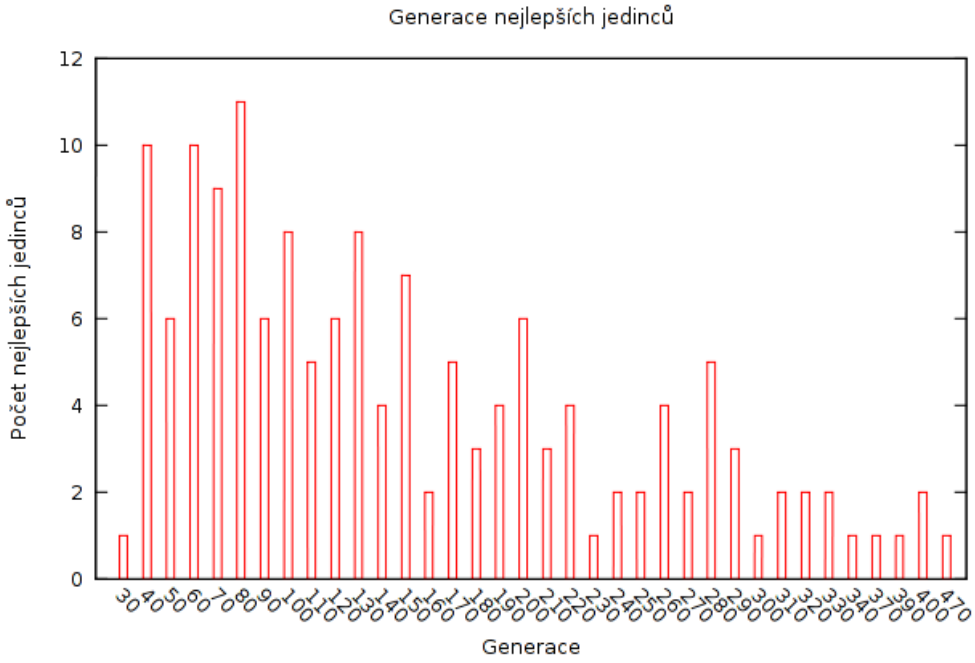
Dosažené výsledky

Po dokončení implementace bylo se systémem provedeno mnoho experimentů zaměřených na zjištění optimálních parametrů optimalizace. Experimenty probíhaly s několika různě početnými náhodnými množinami turnusů. Při experimentováni byly měněny hodnoty parametrů a po dokončení optimalizace byla dále porovnávána kvalita výsledků. Hodnocení kvality bylo založeno pouze na procentuální úspěšnosti optimalizace a délce optimalizace. Vzhledem k tomu, že testovací množiny byly náhodné, nejsou pro ně známy optimální systémy turnusů. Pro zjištění nejpravděpodobnějšího optimálního systému turnusů bylo provedeno s každou množinou více než tisíc běhů s nejrůznějšími hodnotami parametrů. Optimem byl poté určen nejlepší výsledek ze všech běhů. Tak velký počet běhů byl možný pouze masivní paralelizací výpočtů pomocí spuštění systému na přibližně jednom stu školních PC.

Při experimentování s vytvořeným systém bylo zjištěno, že algoritmus optimalizace často uvázne v lokálním extrému. S větším počtem spojů se tento jev zesiluje. Pro úspěšnou optimalizaci, která nalezne globální extrém, je nutné provést spuštění optimalizace několikrát a jako výsledek použít nejlepší turnusy ze všech běhů. Díky několikerému spuštění optimalizačního procesu je možné nežádoucí jev úplně odstranit.

Experimenty ukázaly, že lokální vyhledávání je pro malé množiny spojů naprosto zbytečné. Lokální vyhledávání turnusů má za cíl upravit indexy definující hranice turnusů tak, aby došlo například ke sjednocení dvou turnusů do jednoho nebo přesunutí nevhodného spoje z konce jednoho turnusu na začátek druhého turnusu. Při experimentech nebyly operace provedené lokálním vyhledáváním nikdy uplatněny a čas strávený touto činností byl využit naprosto zbytečně. Tato vlastnost lokálního vyhledávání byla zjištěna pouze na malých množinách spojů. V praxi bývají turnusy vytvářeny pro několikrát větší množiny spojů a tam se lokální vyhledávání může uplatnit.

Experimenty byly prováděny se třemi množinami spojů. Velikosti použitých množin byly 20, 40 a 60 spojů. Pouze pro množiny velikosti 20 a 40 spojů bylo nalezeno optimum a bylo tak možné určit parametry, které mají 100% úspěšnost optimalizace. U množiny spojů nebylo nalezeno optimum a tak byla určena pouze maximální generace, při které byl nalezen



Obrázek 8.1: Histogram četností nejlepšího jedince v generaci.

Tabulka 8.1: Tabulka parametrů a výsledků experimentů.

Spojů	Generací	Populace	Celkový čas	Počet běhů
20	500	2000	10 minut	5
40	700	4000	7 hodin	40
60	4000	12000	18 hodin	10

nejlepší výsledek. Výsledky s hodnotami specifickými pro rozdílné množiny jsou uvedeny v tabulce 8.1. Ostatní parametry byly u všech množin stejné a jsou uvedeny v tabulce 8.2. Tyto parametry byly zvoleny na základě výsledků experimentů. S jejich použitím bylo dosaženo největší míry úspěšnosti u množin s 20 a 40 spoji. Lze je považovat za nejlepší parametry a tedy i výsledek experimentů.

Velikost populace byla stanovena relativně k počtu jedinců v populaci. Se zvětšující se populací dochází dříve k lokální uváznutí a je možné snížit počet generací. S menším počtem generací trvá výpočet kratší dobu a je možné provést více nezávislých běhů optimalizačního algoritmu. Hodnota počtu generací byla stanovena podle nejvyššího čísla generace, ve které byl nalezen nejlepší jedinec. V histogramu na obrázku 8.1 jsou na ose X čísla generací, ve kterých byl nalezen výsledný jedinec, a hodnoty na ose Y udávají četnost výskytu nejlepšího jedince v dané generaci. Pro zjednodušení jsou generace zaokrouhleny na desítky. Histogram je vytvořen z výsledků optimalizace dvaceti spojů při dvoutisících jedincích v populaci.

Po experimentech pro určení parametrů bylo provedeno několik dalších se speciální

Tabulka 8.2: Tabulka vhodných parametrů.

P. křížení	P. mutace	Nahrazovaná populace
0,4	0,9	70%

množinou spojů. Výjimečnost byla v tom, že všechny spoje bylo možné umístit do jednoho turnusu. Množina obsahovala 108 spojů a optimálním řešením je uspořádání spojů podle času odjezdu. Výsledky optimalizace ukázaly rezervy systému. Nejlepší výsledek obsahoval 10 turnusů, což je velmi špatné řešení. Ovšem při řešení tohoto problému nedošlo k uváznutí v lokálním extrému. Při experimentech byl zvětšován počet generací od 1000 do 100000 a se zvětšováním počtu generací se výsledky nepatrně zlepšovaly. Populace obsahovalo 2000 jedinců a výpočet trval 13 hodin. Je pravděpodobné, že zvětšením počtu generací a jedinců by bylo dosaženo lepších výsledků.

S množinou obsahující jeden turnus byl proveden ještě jeden experiment. Tentokrát množina obsahovala pouze 18 spojů. V tomto případě byl algoritmus schopen nalézt optimální řešení s pravděpodobností 0,45. V ostatních případech byly nalezeny různé varianty dvou turnusů. Se správně nastavenými parametry nebyly výsledky nikdy horší než dva turnusy.

Kapitola 9

Závěr

V práci byl navržen princip činnosti algoritmu optimalizace turnusů jízdních řádů s pomocí memetického algoritmu. Navržený algoritmus byl dále implementován v systému pro tvorbu optimálních turnusů jízdních řádů. Z dosažených výsledků jasně plyne, že algoritmus je schopný nalézt optimální řešení problému.

Z výsledků experimentů také vyplývají další možnosti vývoje. Nejdůležitějším cílem dalšího vývoje je zvýšení úspěšnosti jednotlivých běhů optimalizačního algoritmu. Zvýšením úspěšnosti je možné omezit počet běhů potřebných pro nalezení optima. Vylepšení úspěšnosti by mohlo být dosaženo aplikací sofistikovanějšího modelu genetického algoritmu. Například využitím ostrovního modelu by mohlo být dosaženo menší pravděpodobnosti uváznutí v lokálním extrému. Dalším cílem tak může být zlepšení možnosti paralelizace výpočtu. Použitá knihovna GAlib není implementována jako bezpečná ve vláknech a spuštění více vláken s genetickým algoritmem je nyní vyloučeno. Jedinou stávající možností paralelizace je spuštění několika procesů a následné ruční zpracování výsledků.

Jízdní řády se během let příliš neměnily a lidé měli možnost na jejich optimalizaci pracovat po velmi dlouhou dobu. Z tohoto poznatku vyplývá poslední velmi užitečné rozšíření systému. Nyní je algoritmus optimalizace inicializován náhodnou permutací množiny spojů. Úpravou algoritmu, která by umožnila inicializaci pomocí předem stanovené permutace, by bylo možné inicializovat algoritmus turnusy vytvářenými a optimalizovanými lidmi po třicet let. Algoritmus by tak měl velmi výhodnou pozici a každé řešení, které by našel, by bylo vždy lepší než stávající i když by nebylo přímo optimální.

V aktuální podobě je systém prakticky použitelný pro tvorbu nových turnusů i pro optimalizaci stávajících. I kdyby systém nebyl schopen nalézt optimální systém turnusů, je schopný vytvořit nový systém turnusů v podstatně kratším čase než člověk. V práci bylo experimentálně ověřeno, že je možné nalézt optimální řešení, čímž je splněn hlavní cíl této práce.

Popis optimalizačního algoritmu a výsledky práce byly prezentovány na studentské soutěži EEICT. Práce se umístila na druhém místě v magisterském oboru Inteligentní systémy.

Literatura

[1] Andrei Alexandrescu. *Moderní programování v C++*. Computer Press, 2004. ISBN 80-251-0370-6.

[2] Jan Beránek. Analýza a návrh systému pro řízení dopravní společnosti. Master's thesis, Mendelova zemědělská a lesnická univerzita v Brně, Provozně ekonomická fakulta, 2008.

[3] Petr Cenek, Valenta Klima, and Jaroslav Janáček. *Optimalizace dopravních a spojových procesů*. Vysoká škola dopravy a spojů v Žiline. Edičné stredisko VŠDS, 1994. ISBN 80-7100-197-X.

[4] Jakub Filák. Evolutionary optimization of tour timetables. In *Proceedings of the 15th Conference STUDENT EEICT 2009*, volume 2, pages 240–242, 2009.

[5] Sutter Herb and Alexandrescu Andrei. *C++ 101 programovacích technik*. Zonner Press, 2006. ISBN 80-86815-28-5.

[6] Vladimír Kvasnička, Jiří Pospíchal, and Peter Tiňo. *Evolučné algoritmy*. Tlač Vydavateľtvo pri STU Bratislava, 2000. ISBN 80-227-2377-5.

[7] Steve McConnell. *Dokonalý kód*. Computer Press, 2006. ISBN 80-251-0849-X.

[8] Rudolf Pecinovský. *Návrhové vzory*. Computer Press, 2007. ISBN 978-80-251-1582-4.

[9] Josef Schwarz and Lukáš Sekanina. *Aplikované evoluční algoritmy EVO, Studijní opora*. verze 1.2006.

[10] WWW stránky. Boost c++ libraries. `http://www.boost.org/`. [navštíveno 23. 05. 2008].

[11] WWW stránky. Galib: Matthew's genetic algorithm library. `http://lancet.mit.edu/ga/`. [navštíveno 23. 05. 2008].

[12] WWW stránky. Memetics algorithms' home page. `http://www.densis.fee.unicamp.br/ moscato/memetic_home.html`. [navštíveno 28. 12. 2008].

[13] Wikipedie. Jízdní řád — wikipedie: Otevřená encyklopedie. `http://cs.wikipedia.org/w/index.php?title=Jid=3311939`, 2008. [Online; navštíveno 17. 12. 2008].

[14] Ivan Zelinka. *Umělá inteligence v problémech globální optimalizace.* BEN - technická literatura, 2002. ISBN 80-7300-069-5.

[15] Jan Černý and Pavol Kluvánek. *Základy matematickej teórie dopravy.* VEDA vydavatelstvo Slovenskej akadémie vied, Bratislava, 1991. ISBN 80-224-0099-8.

Příloha A

Parametry systému

Tabulka A.1: Parametry v konfiguračním souboru

Jméno	Rozsah	Výchozí	Popis
population	$< 0, UMAX >$	-	velikost populace
generations	$< 0.0, Inf >$	-	konstanta výpočtu počtu generací
crossover	$< 0.0, 1.0 >$	-	pravděpodobnost křížení
mutation	$< 0.0, 1.0 >$	-	pravděpodobnost mutace
replacement	$< 0.0, 1.0 >$	-	velikost nahrazované populace
tabusteps	$< 0, UMAX >$	-	počet kroků zakázaného hledání
tabusetsize	$< 0, UMAX >$	-	velikost tabu množiny
scheduletreshold	$< 0.0, Inf >$	-	práh vhodnosti turnusu
schedulebestduration	$< 0, UMAX >$	585	nejlepší délka turnusu v sekundách
schedulemaxduration	$< 0, UMAX >$	690	maximální délka turnusu v sekundách
scheduleminwaytingtime	$< 0, UMAX >$	45	délka přestávky v sekundách
scheduleemptytimecost	$< 0, UMAX >$	4	cena prázdného přejezdu
schedulelongertimecost	$< 0, UMAX >$	1	cena za delší turnus
scheduleshortertimecost	$< 0, UMAX >$	2	cena za kratší turnus
schedulewaytingtimecost	$< 0, UMAX >$	2	cena za čekání
distanceunitprice	$< 0, UMAX >$	40	cena ujeté jednotky vzdálenosti
vehicleunitprice	$< 0, UMAX >$	25000	cena za jeden dopravní prostředek
waitingunitprice	$< 0, UMAX >$	20	cena za čekání

Tabulka A.2: Parametry příkazové řádky

Jméno	Rozsah	Výchozí	Popis
h, help	-	-	pro získání nápovědy
i, interactive	-	-	systém komunikuje s uživatelem
f, force	-	-	spustí optimalizaci bez kontroly parametrů
s, silent	-	-	pro zakázání informačních výstupů
w, write-result	-	-	pro výpis výsledků na standardní výstup
o, output	soubor	-	jméno soubor pro uložení výsledků
r, rounds	$< 1, UMAX >$	3	počet běhů optimalizace
b, begin	datum	aktuální	první den optimalizace
e, end	datum	aktuální	poslední den optimalizace
c, config-directory	adresář	aktuální	adresář s konfiguračním souborem

Použité zkratky

UMAX je konstanta odpovídající maximální hodnotě typu `unsigned int`

Inf je konstanta odpovídající plus nekonečnu